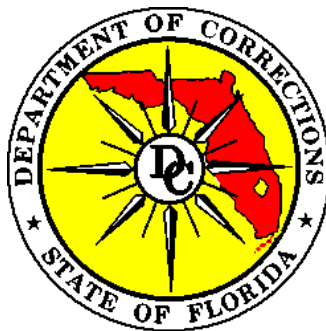


Correctional Operations Trend Analysis System (COTAS)

Final Report

Florida Department of Corrections



Contents

Executive summary	1
Purpose	1
Goals	1
Application overview	2
Application architecture	3
Data warehouse	4
Technology overview	5
Table definitions.....	6
Extract, transform and load processes.....	7
Technology used	7
General business rules for all packages	7
Data sources.....	7
Environment.....	8
Packages developed	9
Main master package	10
Dim_Facility.....	11
Disaster recovery	12
Planning and designing for disaster recovery	12
Selecting a recovery model	12
Cube	16
DCDW structure	16
DCDW data source view.....	16
Cube design.....	17
Cube measures.....	17
Custom calculations	17
Reports.....	18
Analytical reports	18
Maintenance reports	20
General business rules	20
Prediction reports	24
Auditing reports	26
Configuring Dundas gauges.....	26
Configuring Dundas charts	30
Configuring Dundas graphs	32
Environment	34
Server name	34
Reporting services file path.....	34
Report source code location	34
Configuration	34
Custom software components	34
Data mining.....	35
Training methodology	35
Predictions	38
Accuracy.....	39
Personnel	42
Glossary.....	43
Department terms	43
Technical terms.....	44
Appendix	45
Appendix A - Packages Developed	45
Dim_DormBed.....	50
Dim_Gang	55

Dim_DrugTest	60
Dim_Offender	64
StaffDetail	70
Dim_ExternalMovements	75
Dim_HighProfile_Offender	81
Dim_Health_Offender.....	88
Dim_ManagementScores_Offender	95
Dim_Investigations	101
GrievanceOfficeDetail	106
GrievanceFieldDetail.....	110
Dim_DisciplinaryReports.....	115
UOFDetail.....	120
Dim_Escape.....	125
Fact_Event	129
Fact_FacilityActiveInmates	136
DataMiningFacility	139
DataMiningInmates	145
ProcessCube.....	151
Appendix B – COTAS Tutorial	161
Identifying Issues before they develop	161
Conducting Historical Analysis	162
Investigating events of interest	162
Monitoring facilities (chief security officers)	162
Reviewing facility events (wardens).....	164
Drilling into event details (operational staff).....	165
Predicting and mitigating risk	167
Assessing risk with the Inmate Predictor	167
Evaluating risk factors with the Facility Predictor	169

Executive summary

Purpose

The Florida Department of Corrections (FDOC) was awarded a grant by the National Institute of Justice in September, 2006 for the development of the Correctional Operations Trend Analysis System (COTAS). The purpose of this application is to create a correctional crime mapping and information management system that monitors cross-functional operations and can be quickly viewed by all levels of correctional management. The target audience for this application is FDOC personnel, primarily Warden and Duty Warden Staff at the individual institutions.

FDOC needed an application that would enable corrections personnel to analyze large amounts of violent event data. In addition, it needed the ability to use inmate and facility attributes to identify when characteristics previously associated with violence are converging to form potentially dangerous conditions. Finally, FDOC needed to identify trends, patterns and areas of concern within specific units, institutions or statewide. To meet these specifications, Idea collaborated with FDOC in the development of COTAS.

Goals

This document contains technical documentation for the COTAS project developed by Idea's consultants. Its goals are to provide support for the COTAS application on an ongoing basis and serve as a reference point in the event it becomes necessary to restore the application setup. Finally, Idea worked with FDOC to define and implement the goals and objectives listed in Table 1.

Table 1: Goals and objectives		
Number	Goal	Objective
1	Develop a systems approach to disorder analysis that enhances long-term safety and security	Reduce officer and inmate injuries
2	Integrate COTAS into management decisions to supervise inmates efficiently, effectively and safely	Reduce positive drug tests to increase the ratio between inmate assignments and assessed need
3	Use COTAS to review operational indicators of training needs to maintain and increase staff professionalism	Reduce staff turnover

Application overview

COTAS provides information in highly summarized graphical dashboards and detailed investigation reports. In addition, it provides links to existing Web-based systems to avoid the creation of duplicate reports. COTAS provides views of event information in a variety of ways so prison officials can easily recognize trends and other patterns to deal with them as early as possible. COTAS also provides predictive scoring that uses advanced predictive analytical methods. With this functionality, corrections personnel can review inmate histories and the history of similar peers to determine which inmates have the highest potential for committing a violent event in the next thirty days. Finally, COTAS provides predictive scoring for facilities based on the relative amount of gang activity at facilities.

More detailed information about the screens and data presentation of COTAS is in the user guide for COTAS in Appendix B – COTAS tutorial.

Application architecture

Figure 1 shows the components of the COTAS application architecture. In this implementation, separate servers were used for the relational databases: SQL Server, the Cubes, the OLAP server and the Report Server for the reporting portal. The Microsoft SQL Server platform includes each of these components. It is also flexible and allows multiple architectural possibilities ranging from everything on a single server to multiple servers for each component. The specific architecture depends on the number of users the system will handle and the degree of failsafe operation desired.

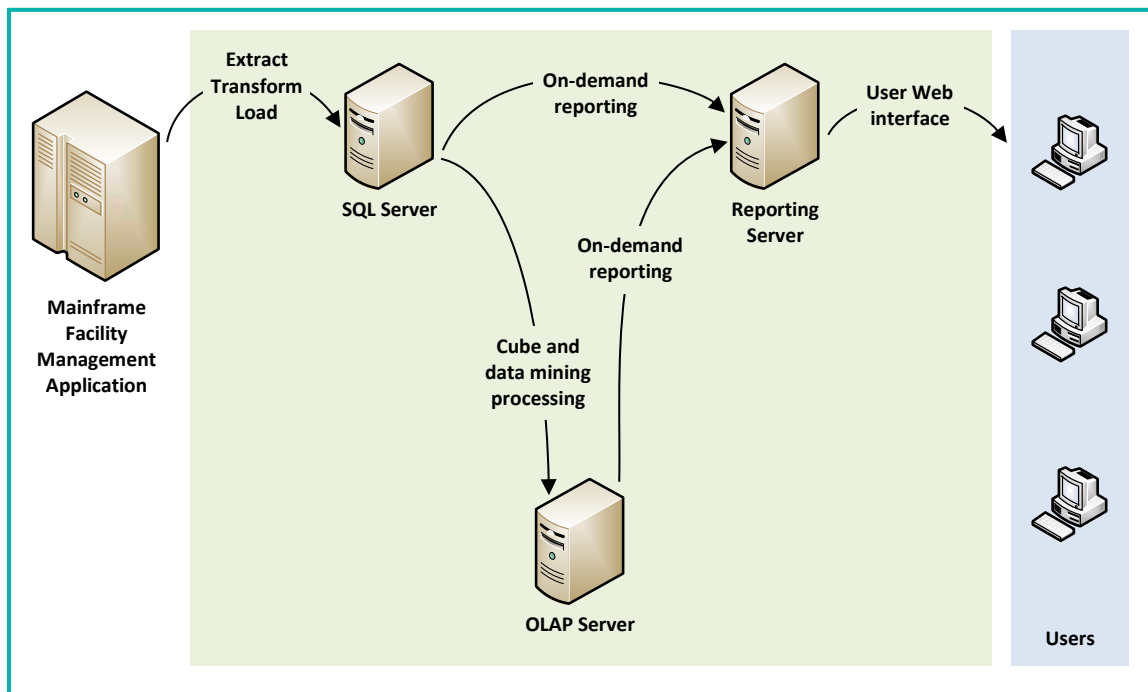


Figure 1: COTAS application architecture

The flexibility to balance budgets against benefits can accommodate any size organization. Businesses often try a prototype before scaling up and rolling it out as an enterprise application. More details about architecture are provided in the *COTAS Installation Manual*.

Note: See *Microsoft Books Online* for thorough information about architecting the Microsoft Business Intelligence components packaged in its SQL Server product.

COTAS gathers data from an existing inmate and facility management mainframe system. In this case, flat files are exported to a file share where they are picked up and placed into a SQL Server staging database. SQL Server Integration Services (SSIS) packages that process the flat files are not mentioned further in this document because they would be specific to each implementation of the application. The staging database is provided since the tables are the starting point for the COTAS SSIS packages.

COTAS processes new data nightly through a series of scheduled jobs running SSIS packages that pull data from the staging database and place it into the data warehouse. Other SSIS packages later in the series process the Analysis Services cube and the data mining model. This document describes all of these packages in detail later.

The Report Server consists of a Web portal and a couple of databases used for permanent storage of the reports and cached temporary data. Reports are developed using an application included with the SQL Server product named Business Intelligence Development Studio (BIDS). Once developed, they are deployed to the Web portal for use. Either the Active Directory (AD) group-based or individual-based security can be applied to entire folders of reports or individual reports to control viewing permissions. Reports can draw from the data warehouse tables directly, pre-summarized data in the cube, the output of the data mining model or a combination of the three.

The availability of data points depends on the implementation of COTAS and the information in the existing inmate facility management system. Some data points shown in this implementation may not be available for other implementations while other implementations may have data points not shown here. You can customize COTAS by adding variables to the data loads and mining model.

Data warehouse

As it applies to this document and project, the data warehouse is the actual database used as a central repository to hold historic information in an integrated and consistent format.

Technology overview

The data warehouse was constructed with Microsoft SQL Server 2005 sp2 and stored in the Department of Corrections Data Warehouse (DCDW). As shown in Figure 2, the data warehouse contains dimensional tables and fact tables among others for the cube's data model. The dimensional tables are denormalized to allow for faster reporting of data. The dimensions store entities such as offender and facility; the fact table holds measures such as events that have occurred.

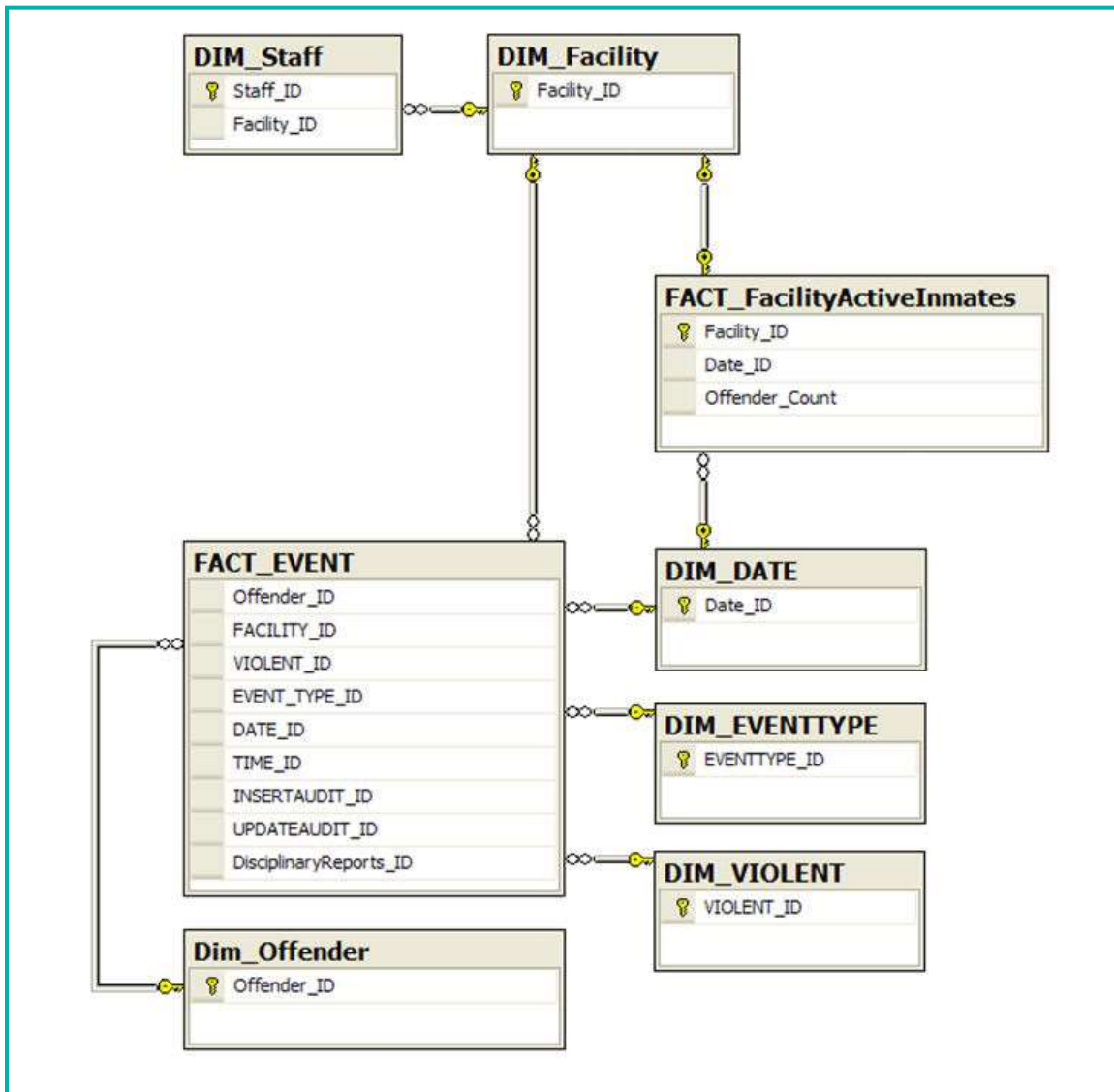


Figure 2: Dimensional tables and fact tables in Department of Corrections Data Warehouse (DCDW)

Table definitions

Dimensions

DIM_Staff	Holds historical information specific to security officers such as date of birth, name and current training status (Current records can be filtered with a null EndDate. Specific points in time are indicated between the startdate and enddate.)
DIM_Facility	Holds historical information about facilities (Current records can be filtered with a null EndDate. Specific points in time are indicated between the startdate and enddate.)
DIM_Date	Contains a daily date entry for every date between 1/1/1901 and 12/31/2100 (This dimension allows for quick joins to segment data by certain date groups such as month and quarter.)
DIM_EVENTTYPE	Stores a look up to an event description such as escape and grievance
DIM_Violent	Describes violent and nonviolent events
DIM_Offender	Holds historical information specific to offenders. Information such as custody status and date of birth (Current records can be filtered with a null EndDate. Specific points in time are indicated between the startdate and enddate. For active inmates, use only inmate status='A'.)

Facts

FACT_EVENT	The main fact table for the COTAS project, which holds all event information and relates to appropriate dimensions (This table maintains a running history of all events recorded and is stored by date of occurrence.)
FACT_FacilityActiveInmates	Allows for fast lookup (pre-aggregated) of the inmate population on a certain day

Extract, transform and load processes

Technology used

Idea implemented COTAS with SQL Server Integration Services (SSIS) 2005.

General business rules for all packages

Package auditing

All packages should include auditing to indicate the following information:

- When the package was started
- When the package was completed
- Whether the package was successful
- How many rows were inserted, updated or deleted

Data extraction

A variable is set in every package by a stored procedure (audit.up_Update_Time_Set) to bring back only data from that date and after. This stored procedure looks at the package auditing table (audit.ExecutionLog) in the Audit_DB database and brings back the date from the last successful completion of the package.

Slowly changing dimensions

Slowly changing dimensions preserve old values and add new records when values change. Most of the packages include coding that uses the lookup transforms in SSIS as the mechanism for handling slowly changing dimensions. This approach ensures that only needed data is inserted into tables.

Easily configured packages

All packages use an extensible markup language (XML) configuration file found in a shared location on all servers: C:\config\SSIS. This configuration file should tell each package which server locations to use during data extraction.

Data sources

Figure 3 shows the source servers and databases that feed into COTAS.

Figure 3: COTAS data sources

Environment

Server name

Table 2 lists the locations of the production and development servers.

Table 2: Server locations	
Server	Location
	REDACTED

Packages location

Table 3 lists the locations of the production and development packages.

Table 3: Production and development package locations	
Package	Storage
Production	Stored on SQL Server: MSDB\DCDW
Development	Stored on SQL Server: MSDB\DCDW

SSIS source code location

All packages were saved with Microsoft Visual SourceSafe in \$/DCDW/SSIS Code/DOC_SSIS/DOC_SSIS/DOC_SSIS.

Configuration

Table 4 lists the locations of the production and development data sources.

Table 4: Production and development data source locations	
Data source	Location
Production	Shared folder (C:\config\SSIS)
Development	Shared folder (C:\config\SSIS)

Packages developed

Idea developed the following packages for the COTAS application:

- Main
- Dim_Facility
- Dim_DormBed
- Dim_Gang
- Dim_DrugTest
- Dim_Offender
- StaffDetail
- Dim_ExternalMovements
- Dim_HighProfile_Offender
- Dim_Health_Offender
- Dim_ManagmentScores_Offender
- Dim_Investigations
- GrievanceOfficeDetail
- GrievanceFieldDetail
- Dim_DisciplinaryReports
- UOFDetail
- DimEscape
- Fact_Event
- Fact_FacilityActiveInmates
- DataMiningFacility
- DataMiningInmates
- ProcessCube

Main master package

Figure 4 shows the relative running order of the packages in the Main the master package. The Main package controls the flow of the entire extract, transform and load (ETL) process and loads each data warehouse table sequentially.

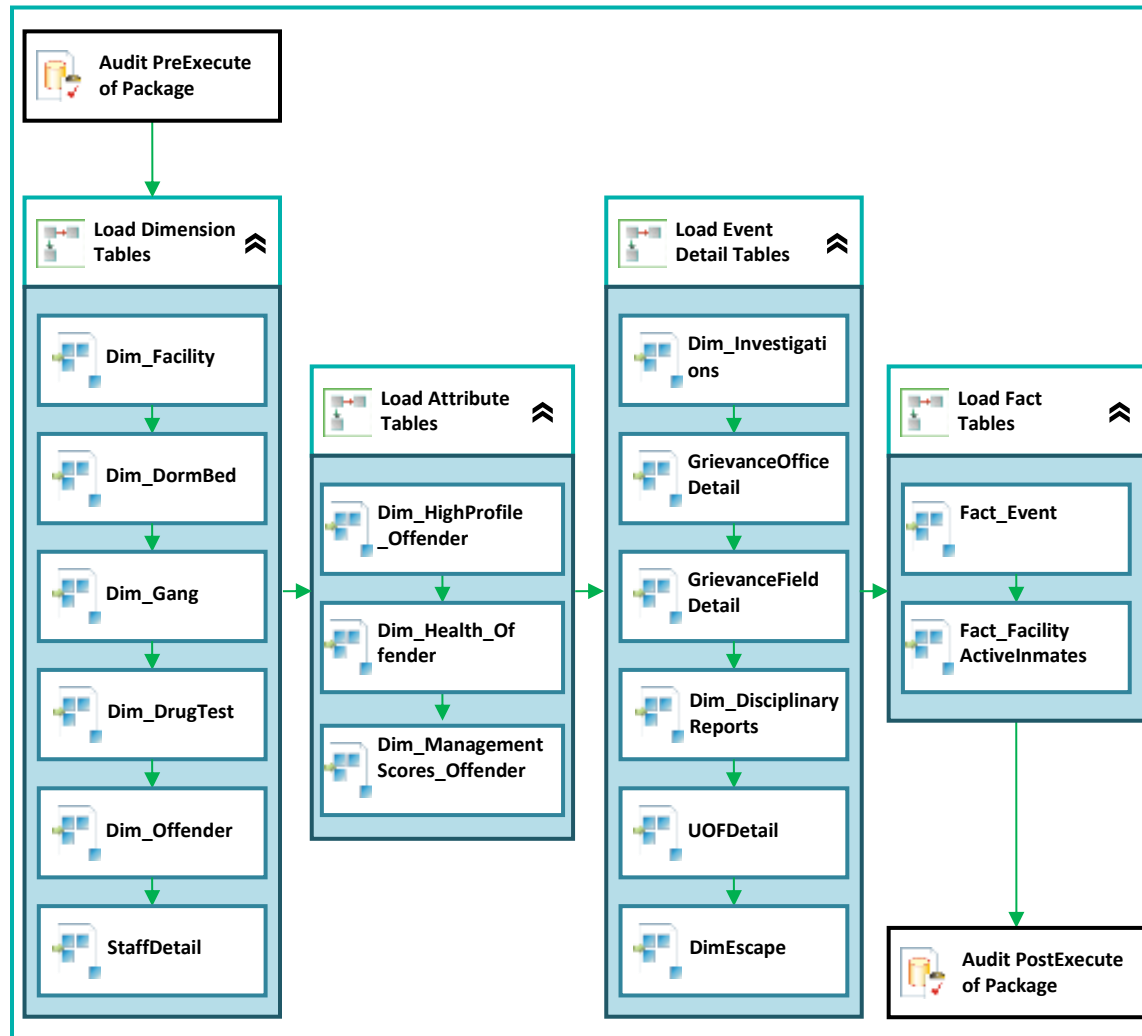


Figure 4: Running order and ETL processes of the COTAS packages in the Main master package

The individual packages in the Main Master Package are outlined in Appendix A – Packages Developed.

Dim_Facility

As shown in Figure 5, the Dim_Facility package updates the table Dim_Facility using the stored procedure usp_Dim_Facility to load the data warehouse database DCDW located on the REDACTEDserver. This stored procedure extracts all the facilities in Florida gathered from the server REDACTEDin the DCIS database. It also adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTEDserver.

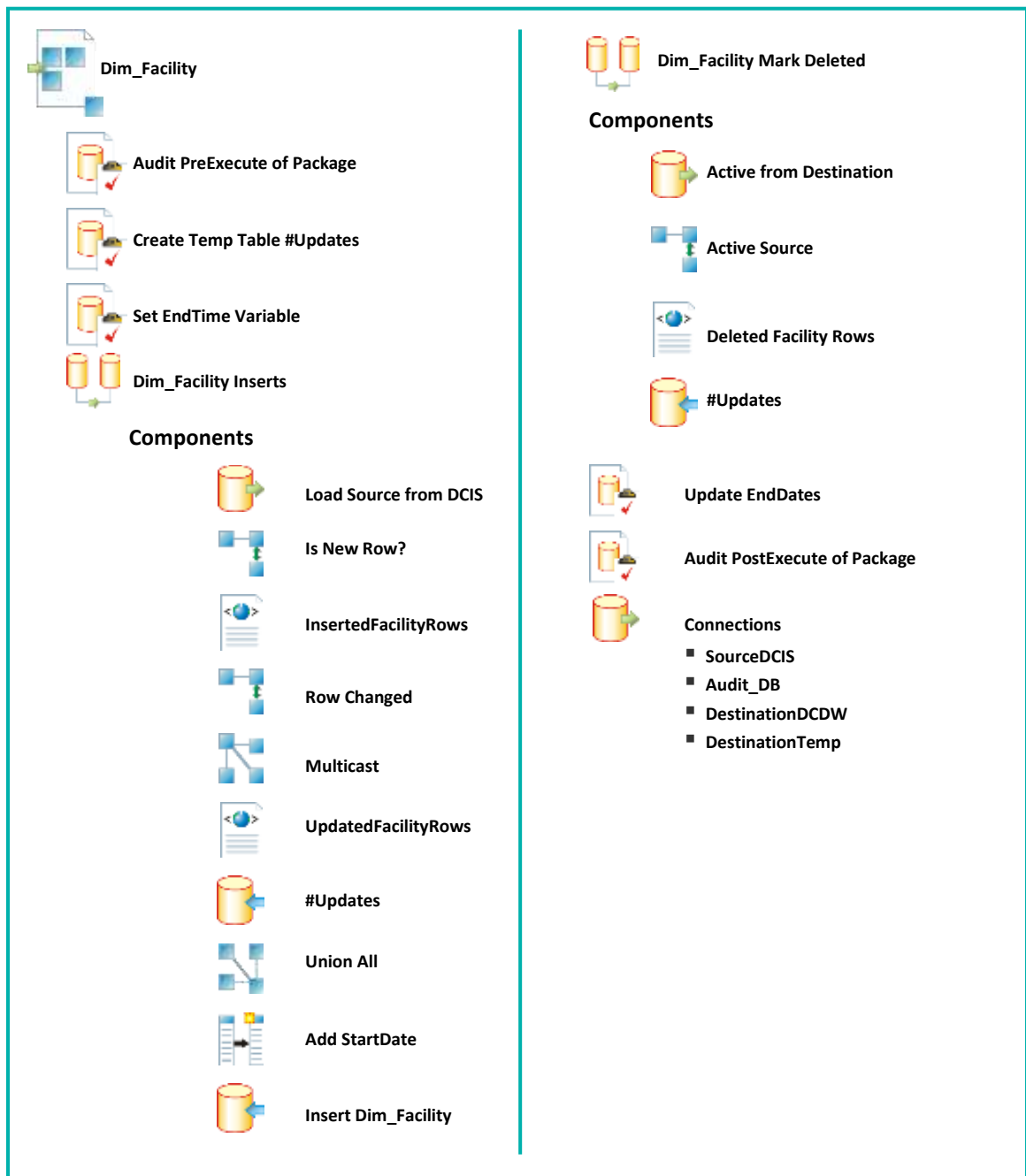


Figure 5: DIM_Facility package

Disaster recovery

Planning and designing for disaster recovery

The purpose of creating SQL Server backups is to help you recover a damaged database; however, backing up and restoring data must be customized to the specific environment and must work with the available resources. Therefore, a reliable use of backup and restore for recovery requires a backup and restore strategy. A well-designed backup and restore strategy maximizes data availability and minimizes data loss, while considering unique business requirements.

A sound strategy contains a backup and restore portion. The backup portion defines the following parameters:

- Type and frequency of backups
- Nature and required hardware speed
- How backups are to be tested
- Where and how backup media is to be stored (including security considerations)

The restore portion defines who is responsible for performing restores and how restores should be performed to meet your goals for availability of the database and for minimizing data loss. Idea recommends documenting your backup and restore procedures and keeping a copy of the documentation in your run book.

Designing an effective backup and restore strategy requires careful planning, implementation and testing. You do not have a backup strategy until you have successfully restored backups in all the combinations that are included in your restore strategy. At a minimum, you must consider the following factors:

- The production goals for your databases, especially for availability and protection of data from loss
- Database attributes such as size, usage patterns, nature of content and data requirements
- Resource constraints such as hardware, personnel, backup media space and the physical security of stored media

Selecting a recovery model

Backup and restore operations occur within the context of a recovery model, which is a database property that controls the management of the transaction. Also, the recovery model of a database determines what types of backups and what restore scenarios are supported for the database. Typically a database uses either the simple recovery model or the full recovery model. The full recovery model can be supplemented by switching to the bulk-logged recovery model before bulk operations.

The best choice of recovery model for the database depends on your business requirements. To avoid transaction log management and simplify backup and restore, use the simple recovery model. Use the full recovery model to minimize work-loss exposure, at the cost of administrative overhead, which is the recommendation for this application.

Selecting the optimal recovery model for each of your databases is an essential part of planning your backup and restore strategy. The choice of recovery model for a database depends somewhat on your availability and recovery requirements. The choice of recovery model, in turn, affects the possibilities for disaster recovery for a database.

Recovery models are designed to control transaction log maintenance. Three recovery models exist: simple, full and bulk-logged. Typically, a database uses the full recovery model or simple recovery model. Table 5 summarizes these recovery models.

Table 5: Recovery models			
Model	Description	Work loss exposure	Recover to point in time
Simple	Does not require log backups (Automatically reclaims log space to keep space requirements small, essentially eliminating the need to manage the transaction log space.)	Changes since the most recent backup are unprotected (In the event of a disaster, those changes must be redone)	Can recover only to the end of a backup
Full	Requires log backups (No work is lost due to a lost or damaged data file. Can recover to an arbitrary point in time. For example, before application or user error).	Normally none (If the tail of the log is damaged, changes since the most recent log backup must be redone.)	Can recover to a specific point in time, assuming that your backups are complete up to that point in time
Bulk logged	Requires log backups (An adjunct of the full recovery model that permits high-performance bulk copy operations. Reduces log space usage by using minimal logging for most bulk operations.)	If the log is damaged or bulk-logged operations occurred since the most recent log backup, changes since that last backup must be redone. Otherwise, no work is lost.	Can recover to the end of any backup (Point-in-time recovery is not supported.)

Designing a backup strategy

After you have selected a recovery model that meets your requirements, you must plan and implement a corresponding backup strategy. The optimal backup strategy depends on a variety of factors. For example, you should ask how many hours a day your applications require database access. If there is a predictable off-peak period, Idea recommends scheduling full database backups for that period. You should also find out how frequently changes and updates are likely to occur. If changes are frequent, consider the following measures:

- Under the simple recovery model, schedule differential backups between full database backups. A differential backup captures only the changes since the last full database backup.
- Under the full recovery model, schedule frequent log backups. Scheduling differential backups between full backups can reduce restore time by reducing the number of log backups you have to restore after restoring the data.

Another part of designing your back up strategy is to ask whether changes are likely to occur in a small or large part of the database. For changes to a large database, in which changes are concentrated in a part of the files or filegroups, partial backups and file backups may be useful. Finally, you should ask how much disk space a full database backup will require. For more information, see *Estimating the size of a full database backup*.

Estimating the size of a full database backup

Before you implement a backup and restore strategy, you should estimate how much disk space a full database backup will use. The backup operation copies your data to a backup file. The backup contains only the actual data in your database and not any unused space. Therefore, the backup is usually smaller than your database. You can estimate the size of a full database backup by using the `sp_spaceused` system stored procedure.

Scheduling backups

After you decide what types of backups you require and how frequently you have to perform each type, we recommend that you schedule regular backups as part of a database maintenance plan.

Testing backups

You do not have a restore strategy until you have tested your backups. It is very important to test your backup strategy thoroughly by restoring a copy of each database onto a test system. You must test restoring every type of backup that you intend to use.

Idea recommends that you maintain an operation manual for each database. The operations manual should document the location of the backups, backup device names (if any) and the amount of time required restoring the test backups.

Managing backup media

At a minimum, Idea recommends that your backup plan include the following provisions for managing backup media:

- A tracking and management plan for storing and recycling backup sets
- A schedule for overwriting backup media
- In a multiserver environment, a decision to use either centralized or distributed backups
- A means of tracking the useful life of media
- A procedure to minimize the effects of a lost backup set or backup media such as the loss of a tape
- A decision to store backup sets on or offsite and an analysis of how it will affect recovery time

Running a base-functionality script

Usually, you include a base-functionality script as part of your disaster recovery plan to confirm that everything works as intended. A base-functionality script provides a dependable tool for the system administrator or database administrator to verify that the database is back in a workable state without depending on end users for verification.

Base-functionality scripts are application specific and take many different forms. For example, on a decision-support or reporting system, the script may just be a copy of several key reporting queries. For an online transaction processing (OLTP) application, the script may execute a batch of stored procedures that execute INSERT, UPDATE and DELETE statements. A base-functionality script might be as simple as a SQL file that sends batched SQL statements to the server from the sqlcmd utility. Another example is using a .bat file that contains both bcp and sqlcmd commands.

Ensuring disaster readiness

To ensure that you are ready for disaster, Idea recommends that you periodically perform several maintenance activities. For example you should test your backup and recovery procedures thoroughly before the occurrence of a real failure. Testing you ensure

- You have the required backups to recover from various failures
- Your procedures are clearly defined and documented
- Any qualified operator can execute your procedures smoothly and quickly

Another way to ensure your disaster readiness is to perform regular database and transaction log backups. Backups minimize the amount of lost data and you should backup system and user databases.

Finally, you should securely maintain system logs. Keep records of all service packs installed on Microsoft Windows and SQL Server. Maintain records of network libraries used and the security mode. Also, if SQL Server is running in Mixed Mode Authentication (SQL Server and Windows Authentication Mode), record the system administrator (SA) password in a secure location.

Cube

DCDW structure

As shown in Figure 6, the data model for the DCDW Cube includes the following tables:

- DIM_Staff
- DIM_Facility
- DIM_VILOENT
- FACT_EVENT
- DIM_DATE
- FACT_GOAL
- Dim_Offender
- FACT_Facility
- DIM_EventType

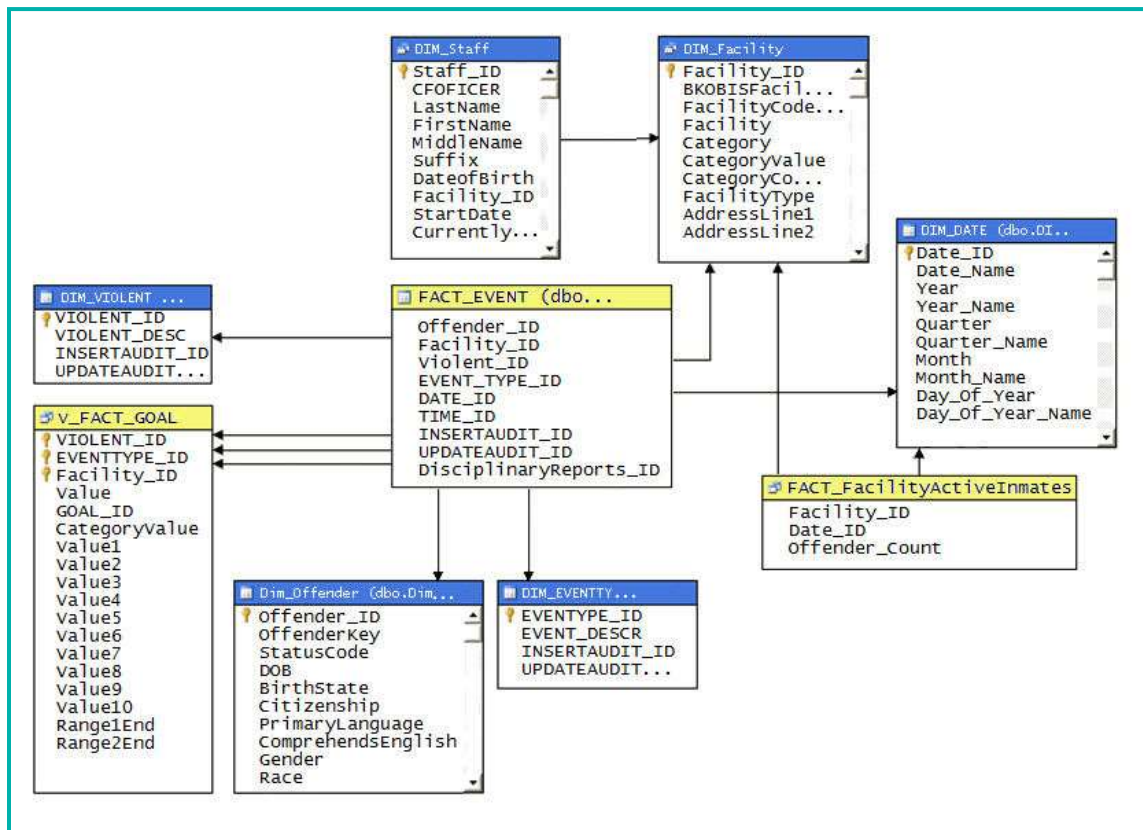


Figure 6: DOC_EVENTS solution in the DCDW cube on the REDACTED server

DCDW data source view

Figure 7 shows the structure of the DCDW data source view.

Figure 7: Data source view name: DCDW

Cube design

The DCDW cube was designed to facilitate quick calculations and aggregations of event-based data over time. A star schema was constructed to preserve the relative simplicity of the cube and keep it easy to maintain.

Note: You can find dimensions and fact information in the data warehouse information section.

Cube measures

GOAL	Used to allow adjusting of scale (needle) in Reports and also the ranges for color schemes
EVENT	Holds event counts
Fact Facility Active Inmates	keeps track of number of inmates over time

Custom calculations

The EventRatePer1000Inmates calculation is used at the facility level to calculate the number of events occurring in the facility per 1000 inmates. The formula for this calculation is $(\text{Events}/\text{\#inmates}) * 1000$.

EventRateScaled	Uses the EventRatePer1000Inmates calc and compares with goal measures to give a scaled value (1-100) based on Goal Table
EventThresholdExceeded	Determines if an event for a specific facility exceeds the <i>red zone</i> threshold, which is defined in the Goal table column Range2 End and pulled into the cube measure range2 end
Sum of Violent Alerts	Calculates the sum of EventThresholdExceeded at the facility level
KPI's events	Holds information to allow Reports to display event information

Reports

Idea used SQL Server Report Server 2005 to create two classes of reports: analysis and auditing.

Analytical reports

Event reports

As shown in Figure 9, a dashboard with historical gauges provides event data by region and consists of the following reports:

- RegionGauges
- RegionGraph
- RegionDashboard
- InmateSearch
- RegionHighProfileInmates
- FacilityHighProfileInmates
- FacilityMonitor
- DevEventSummarybyDorm
- DevEventSummarybyDormEscapes
- EventGraph
- DisciplinaryReport_Investigations
- Inmate Face Sheet URL
- Case_Investigations_Detail
- Use_of_force_Investigations
- Inmate_Grievance_Detail
- Escape Face Sheet URL

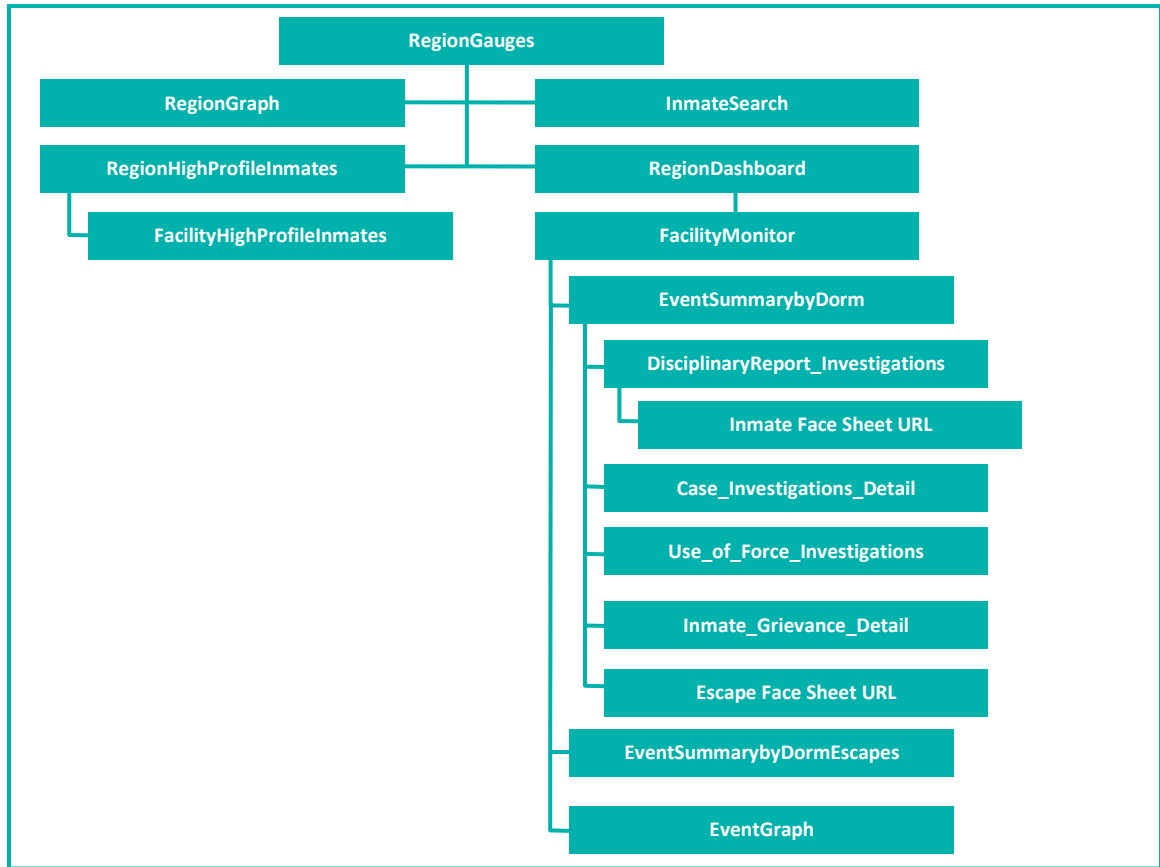


Figure 8: Event Reports workflow

Prediction reports

As shown in Figure 9, a dashboard with predictive gauges provides event data by region and consists of the following reports:

- RegionGauges
- RegionPredictorDashboard
- RegionPredictorList
- PredictorSummaryByDorm
- FacilityPredictorSummaryByDorm
- GangDetail

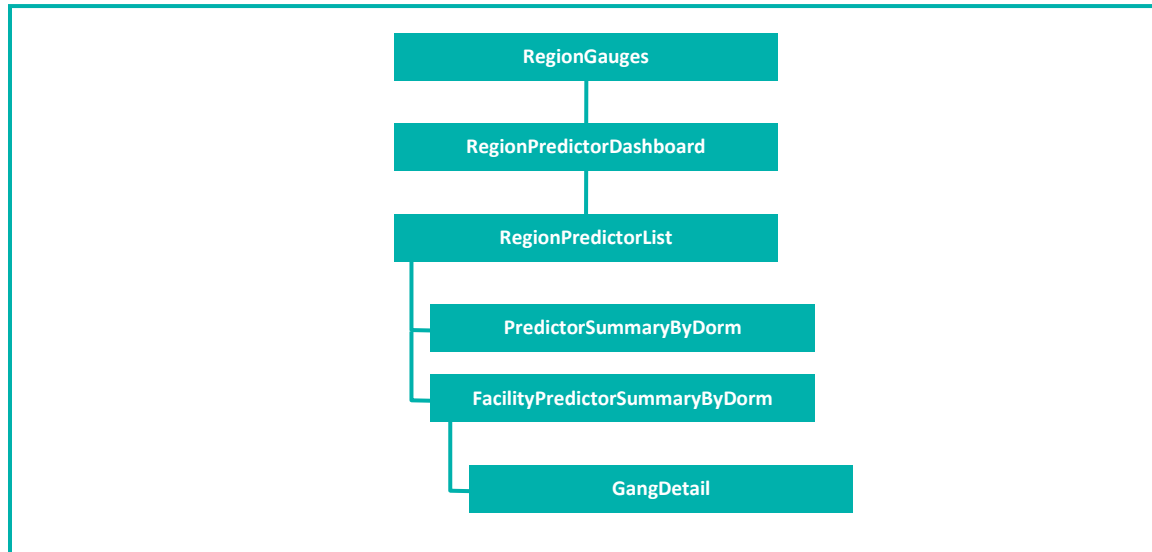


Figure 9: Prediction Reports workflow

Maintenance reports

As shown in Figure 10, the maintenance reports include the AuditPackages report and the AuditSummary report.

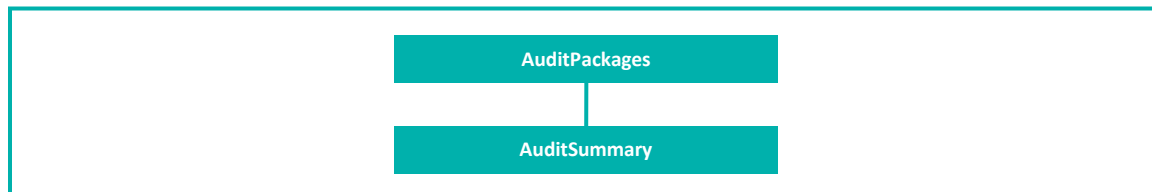


Figure 10: Auditing Reports workflow

General business rules

Table 6 describes the general business rules for all reports.

Table 6: General business rules for all reports	
Number	Rule
1	The target audience for version 1.0 is selected Central Office users
2	Test group should be selected wardens and region administrators
3	A violent event can be either a disciplinary report, escape, investigation or use of force that is recorded as violent
4	A nonviolent event can be either a field grievance, central office grievance, escape, disciplinary report or investigation that is recorded as nonviolent

Table 6: General business rules for all reports

Number	Rule
5	Principles that have been defined though demos: ▪ The Department does not want a map to use GIS for the highest region rollup ▪ The Department prefers gauges ▪ The Department must be able to group summary of event reports by different fields such as Primary Work Assignment, Bed Mission and Description
6	All event reports show a rolling, 30-day event list and counts
7	SSIS packages run at 1:00 a.m. weekdays and do not run on weekends so reports will show a 24-hour latency Monday through Friday

RegionGauges Report

The RegionGauges Report (Figure 11) requires the following features:

- A set of gauges to display the total number of facilities by region that have reached the red threshold
- A set of charts to display the proportional of facilities by region with a high risk of an event
- A snapshot of total events and high profile inmates by region
- A link to Inmate Search Report

Figure 11: RegionGauges Report

Note: For instructions on configuring the gauges, charts and graphs of this report, see the following topics at the end of the Reports section:

- *Configuring Dundas gauges*
- *Configuring Dundas charts*
- *Configuring Dundas graphs*

InmateSearch Report

The InmateSearch Report (Figure 12) requires search capabilities for user-entered Department of Corrections numbers (DCNumber). It must also provide all inmate detail information, including events over the last thirty days.

Figure 12: InmateSearch Report

RegionHighProfileInmates Report

The RegionHighProfileInmates Report (Figure 13) should provide a headcount of all the high-profile inmates in the selected region by facility.

Figure 13: RegionHighProfileInmates Report

FacilityHighProfileInmates Report

The FacilityHighProfileInmates Report (Figure 14) shows all inmates that are considered high profile. This report can be sorted by Bed, Dorm, Type or Disposition and must include comments about the inmate's high-profile status.

Figure 14: FacilityHighProfileInmates Report

RegionGraph Report

The RegionGraph Report (Figure 15) should show trends for violent events for the past year.

Figure 15: RegionGraph Report

RegionDashboard Report

The main table of the RegionGraph Report (Figure 16), should list all facilities in the selected region ordered by red (top), yellow (middle) and green (bottom) status indicators. The main table should also include Category, Facility type and Event columns. The Events column should show the sum total of violent and non-violent events. This report should also provide a picture of the region that links to a Uniform Resource Locator (URL) of a larger image. In addition, the report requires a snapshot of total events for the entire region on RegionGauges. Finally, the last graph must depict the total number of events by category.

Figure 16: RegionDashboard Report

FacilityMonitor Report

The FacilityMonitor Report (Figure 17) requires separate gauges for violent and nonviolent events. The violent gauges should include disciplinary reports, escapes, investigations and use of force (UOF). Nonviolent gauges should include Central Office grievances, disciplinary reports, escapes, field grievances and investigations. Each gauge should show the total number of events for the event type it displays.

The FacilityMonitor gauges should also indicate goals based on thresholds supplied through research. If a gauge touches a new threshold, the status indicator should change. For example, if the needle rests directly between yellow, the gauge should indicate that it has reached red status. Clicking a facility name or individual event, should display an event graph for violent and nonviolent events. Clicking a gauge should display a summary report.

Figure 17: Facility Monitor Report

EventGraph Report

The EventGraph Report (Figure 18) should show events over the last year for the selected facility.

Figure 18: EventGraph Report

EventSummarybyDorm Report

The EventSummarybyDorm Report (Figure 19) should provide a quick description each event and identify it with a log number. It should also group events by bed, bed mission, dorm, primary work assignment and description. Upon the selection of a different group, the sort order should change to the total number of events per group. Finally, clicking on a specific event log number should display detailed information about that event.

Figure 19: EventSummarybyDorm Report

EventSummarybyDormEscapes Report

The EventSummarybyDormEscapes Report (Figure 20) should work exactly the same as the EventSummarybyDorm Report, but the log number links to a URL. For this reason, it is necessary to create a separate report.

Bed	Bed Mission	Dorm	Primary Work Assignment	Description			
Events for Facility: DINSMORE W.R.C.							
Bed :	Bed Mission :	Dorm :	Primary Work Assignment :	Description :	Log Number :	Incident Date :	Incident Time :
				LEVEL ESCAPE	129640	04/05/2008	10:25:00
COTAS Project Implementation Phase 1							5/14/2008 10:31:42 AM

Figure 20: EventSummarybyDormEscapes Report

DisciplinaryReport_Investigations Report

In the DisciplinaryReport_Investigations Report (Figure 21), all detail reports must have a uniform appearance. In addition, images should link to a **Note**: The DCWeb URL is.

Figure 21: DisciplinaryReport_Investigations Report

Inmate Face Sheet URL

The Inmate Face Sheet URL (Figure 22) is found on DCWeb. The URL for this report is

Figure 22: Inmate Face Sheet URL

Case_Investigations_Detail Report

In the Case_Investigations_Detail Report (Figure 23), all detail reports must have a uniform appearance.

Figure 23: Case_Investigations_Detail Report

Use_of_force_Investigations Report

In the Use_of_force_Investigations Report (Figure 24), all detail reports must have a uniform appearance.

Figure 24: Use_of_force_Investigations Report

Inmate_Grievance_Detail Report

In the Inmate_Grievance_Detail Report (Figure 25), all detail reports must have a uniform appearance. The Grievance Report and Central Office Grievance Report should link to this single report.

Figure 25: Inmate_Grievance_Detail Report

Escape Face Sheet URL

For the Escape Face Sheet URL (Figure 26), the Escape detail can be found on DCWeb. Detailed information is only stored on DCWeb fifteen days after the escape or fifteen days after recapture.

Note: The DCWeb URL is

Figure 26: Escape Face Sheet URL

Prediction reports

RegionPredictorDashboard Report

The RegionPredictorDashboard Report (Figure 27) should provide the option to chose between the two predictors (Inmates and Facility).

Figure 27: RegionPredictorDashboard Report

Note: For instructions on configuring charts, see *Configuring Dundas charts* at the end of the *Reports* section.

RegionPredictorList Report

The RegionPredictorList Report (Figure 28) requires a list of facilities in the selected region. In addition, these facilities should be sorted by the highest risk first.

Figure 28: RegionPredictorList Report

PredictorSummaryByDorm Report

The PredictorSummaryByDorm Report (Figure 29) should give a quick description of the inmate and identify each inmate with a risk factor—not a risk percentage. In addition, it should enable grouping by bed, bed mission, dorm and primary work assignment. Finally, clicking specific DC numbers should display detailed inmate information.

Figure 29: PredictorSummaryByDorm Report

Inmate Face Sheet URL

The Inmate Face Sheet URL (Figure 30) is found on DCWeb. The URL for this report is.

Figure 30: Inmate Face Sheet URL

FacilityPredictorSummaryByDorm Report

The FacilityPredictorSummaryByDorm Report (Figure 31) should provide a quick description of the inmates that are suspected or confirmed gang members. Report should be able to group by gang, dorm and primary work assignment. Click on the Gang name a view the gang profile report.

Figure 31: PredictorSummaryByDorm Report

GangDetail Report

The GangDetail Report (Figure 32) should provide a quick profile of the gang in addition to showing its allies and enemies.

Figure 32: GangDetail Report

Auditing reports

AuditPackages Report

The AuditPackages Report (Figure 33) must display all the SSIS packages that are run for the data warehouse. It should also show package runtime and whether it succeeded or failed. Finally, the AuditPackages Report should be grouped by the package name, but needs to show package errors and warnings.

Figure 33: AuditPackages Report

AuditSummary Report

The AuditSummary Report (Figure 34) should display error output from the SSIS packages grouped by missing facility, missing bed or missing offender. Furthermore, each Package ID group should be expandable and provide additional detail about the error output. Finally, the report should be ordered by the time in which the packages were run, starting with the latest.

Figure 34: AuditSummary Report

Configuring Dundas gauges

If the default settings of the Dundas gauges do not meet the specifications of your facility, you can reconfigure them in the GaugeDesigner.

Follow these steps to configure the Dundas gauges:

- 1 Start **Dundas GaugeDesigner**.

Dundas GaugeDesigner appears on your desktop (Figure 35).



Figure 35: Dundas GaugeDesigner

- 2 On the GaugeDesigner **General** tab (Figure 36), click **Data Source** and select the appropriate source.

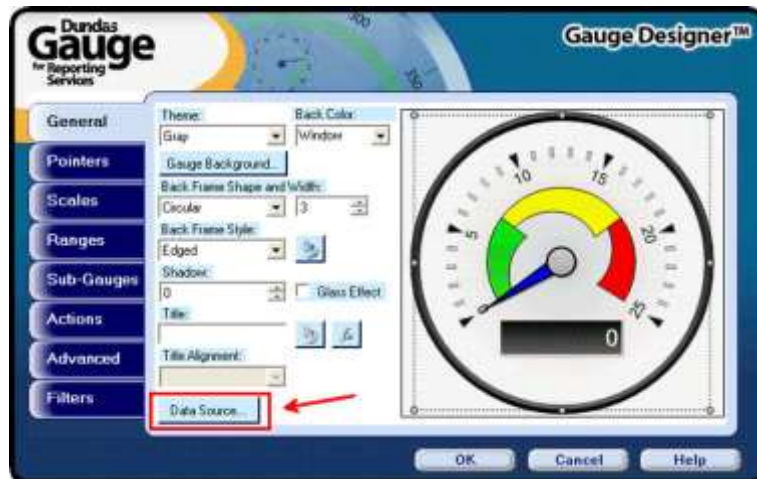


Figure 36: Dundas GaugeDesigner, General tab

- 3 On the **Pointers** tab (Figure 37), click **Function** (f_x) and **Calculate** () to select appropriate pointer value.



Figure 37: Dundas GaugeDesigner, Pointers tab

- 4 On the **Scales** tab (Figure 38), click **Function** (f_x) and **Calculate** () to enter the minimum and maximum scale values in the Minimum and Maximum fields.

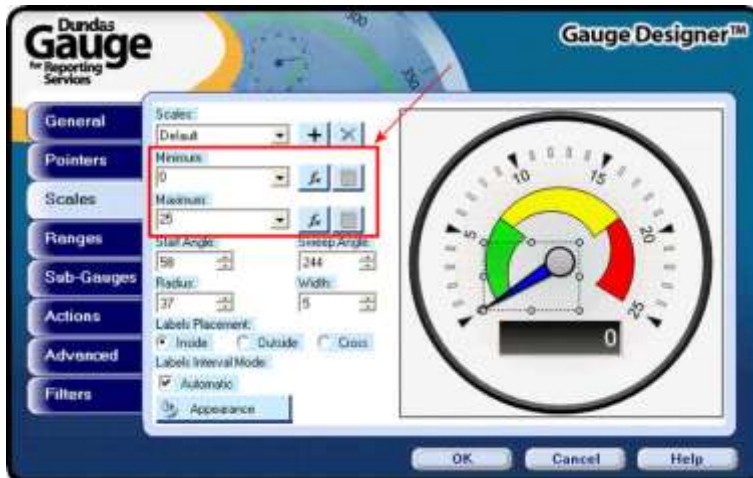


Figure 38: Dundas GaugeDesigner, Scales tab

- 5 On the **Ranges** tab (Figure 39), click **Add** (+), **Delete** (X) to add and delete red, yellow and green ranges.



Figure 39: Dundas GaugeDesigner, Ranges tab

- 6 As shown in Figure 39, click **Function** (f_x) and **Calculate** () to enter the start and end values for the red, yellow and green ranges you added in step 5.
- 7 On the **Sub-Gauges** tab, add a sub-gauge.
The Dundas GaugeDesigner displays the Sub-GaugeDesigner in a separate window.
- 8 On the **General** tab of the Sub-Gauge Designer (Figure 40), click the **Value** drop-down arrow, select **FieldFacilitiesWith**.

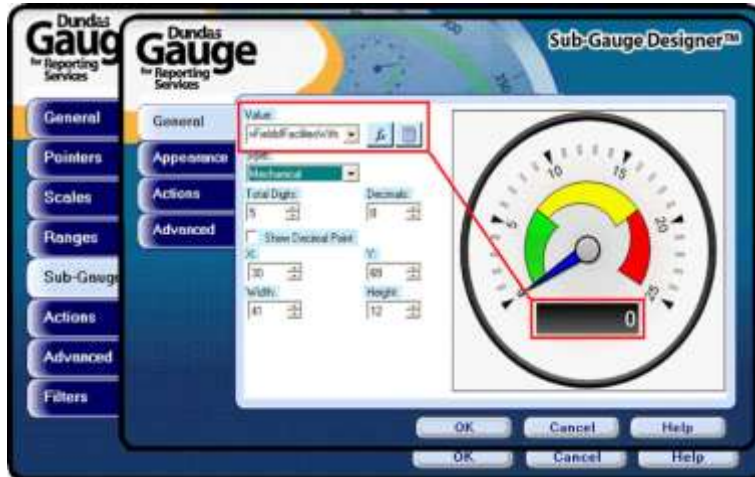


Figure 40: Dundas Sub-Gauge Designer, General tab

- 9 Click **Function** (f_x) and **Calculate** (calculator icon) as shown in Figure 40 to set the sub-gauge counter value.
- 10 Click **OK**.

The Sub-gauge Designer closes and applies your settings to the sub-gauge counter.

- 11 On the **Actions** tab of the GaugeDesigner (Figure 41), click **Parameters** and **Function** (f_x) to set the navigation to the next report.



Figure 41: Dundas GaugeDesigner, Actions tab

Configuring Dundas charts

If the default settings of the Dundas charts do not meet the specifications of your facility, you can reconfigure them in the GaugeDesigner.

Follow these steps to configure the Dundas gauges:

- 1 Start **Dundas ChartDesigner**.

Dundas ChartDesigner appears on your desktop (Figure 42).

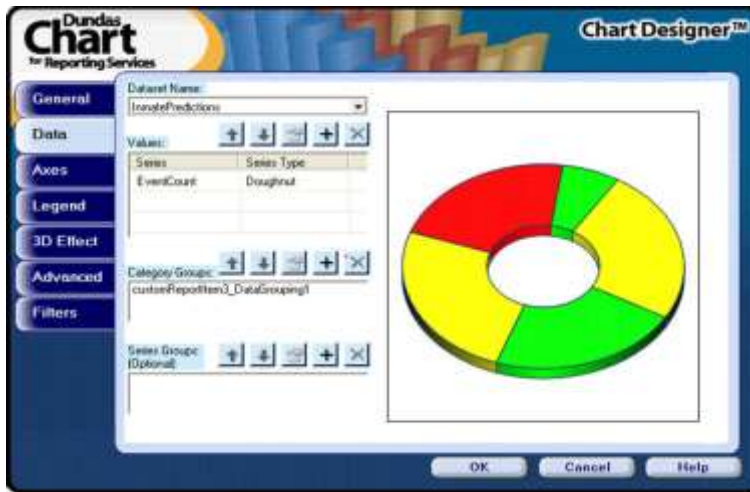


Figure 42: Dundas ChartDesigner

- 2 On the ChartDesigner **Data** tab (Figure 43), click the **Dataset Name** drop-down arrow and select **InmatePredictions**.

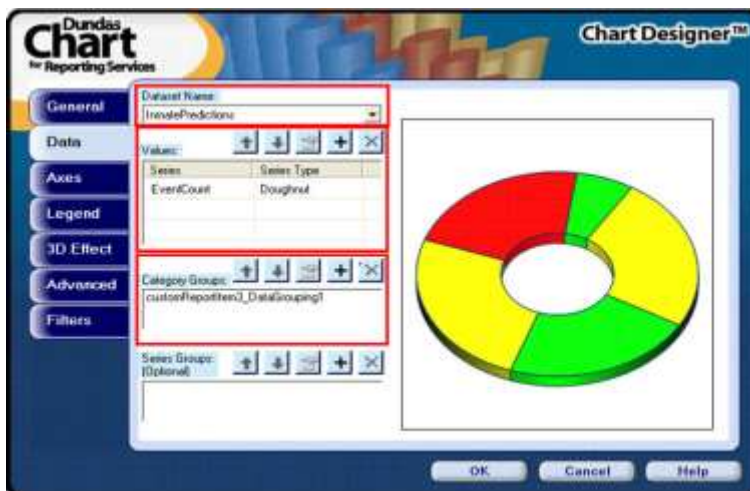


Figure 43: Dundas ChartDesigner, Data tab

- 3 Next to **Values**, click **Properties** (Figure 44).

The Properties dialog box appears.



Figure 44: Properties icon

- 4 Type **EventCount** in the **Series** field and **Doughnut** in the **Series Type** field.
- 5 Set grouping by **Indication** and click **OK**.

The Properties dialog box closes and applies your settings to the chart.

- 6 On the **3D Effect** tab (Figure 45), select the **Enable 3D** check box.

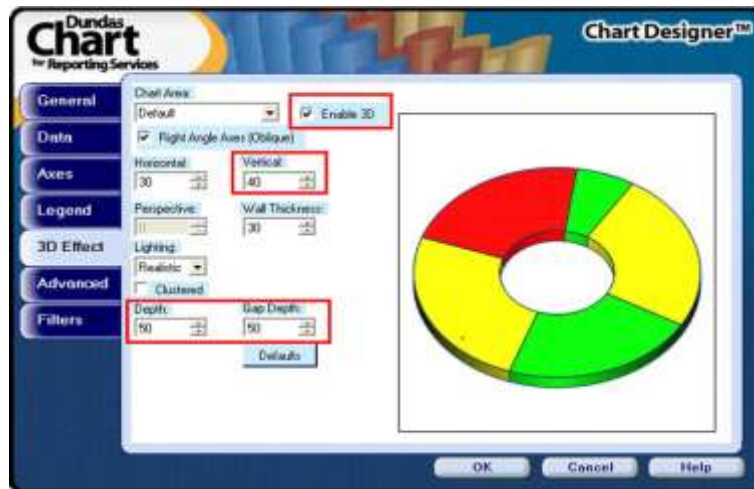


Figure 45: Dundas ChartDesigner, 3D Effect tab

- 7 As shown in Figure 45, scroll the **Vertical** spin box a value of **40**, the **Depth** spin box to a value of **50** and the **Gap Depth** spin box to a value of **50**.
- 8 On the **Filters** tab (Figure 46), set the chart filter by the appropriate region.

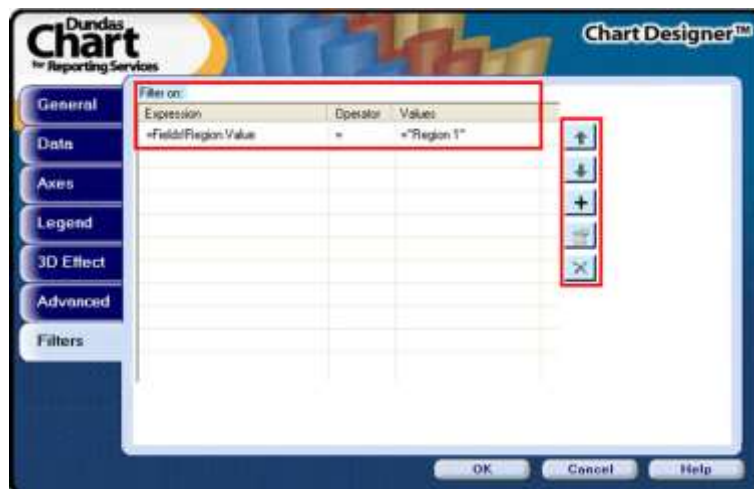


Figure 46: Dundas ChartDesigner, Filters tab

- 9 Click **OK**.

The Chart Designer closes and applies your settings to your charts.

Configuring Dundas graphs

If the default settings for the report graphs do not meet your specifications, you can reconfigure them in the Dundas ChartDesigner.

Follow these steps to configure the Dundas graphs:

- 1 Start **Dundas ChartDesigner**.

Dundas ChartDesigner appears on your desktop (Figure 47).

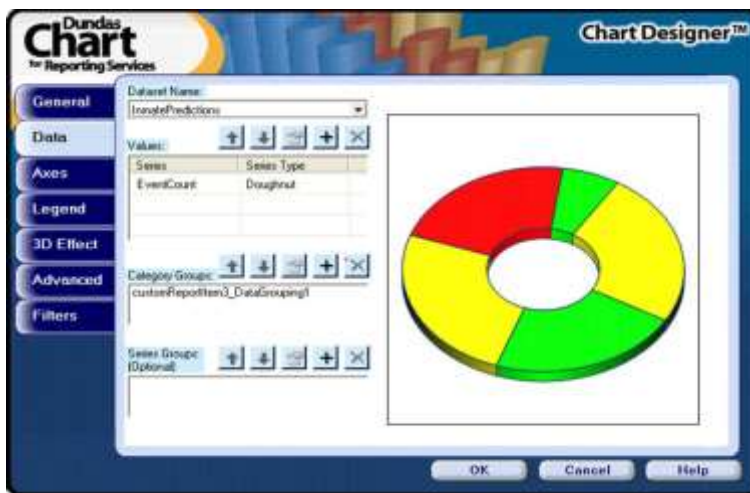


Figure 47: Dundas ChartDesigner

- 2 On the ChartDesigner **Data** tab (Figure 48), click the **Dataset Name** drop-down arrow and select **Category**.

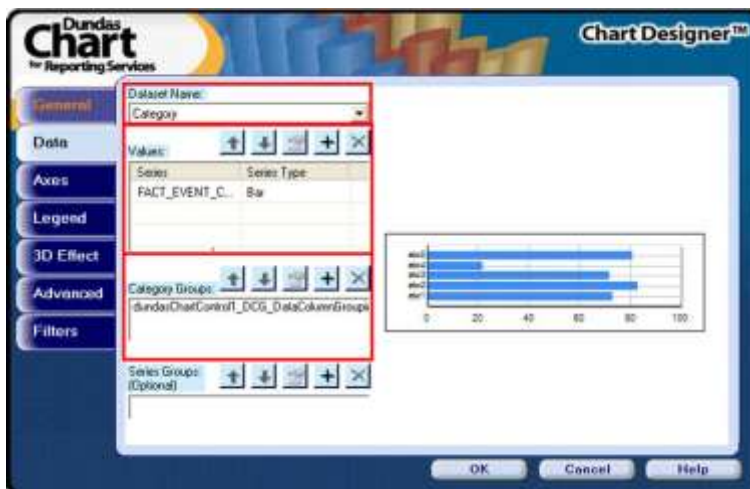


Figure 48: Dundas ChartDesigner, Data tab

- 3 Next to **Values**, click **Properties** (Figure 49).

The Properties dialog box appears.



Figure 49: Properties icon

- 4 Type **Fact_Event_Count** in the **Series** field and **Doughnut** in the **Series Type** field.
- 5 Set grouping by **Category** and click **OK**.

The Properties dialog box closes and applies your settings to the chart.

- 6 Click **OK**.

The Chart Designer closes and applies your settings to your graphs.

Environment

Server name

- Production Environment: REDACTED
- Development Environment: WSCOC4117 (Developer's machine)

Reporting services file path

- Production Environment: G:\Program Files\Microsoft SQL Server\MSSQL.2\Reporting Services
- Development Environment: C:\Program Files\Microsoft SQL Server\MSSQL.3\Reporting Services

Report source code location

All reports were saved in Microsoft Visual SourceSafe
(\$/DCDW/ReportingServicesDevelopment/Report Project 1)

Configuration

Table 7 lists the data sources for the COTAS production and development environments.

Table 7: Data source configuration		
Data source	Production	Development
COTAS_AUDIT	REDACTED	
COTAS_DB	REDACTED	
COTAS_DW	REDACTED	

Custom software components

The following software components were custom developed for COTAS:

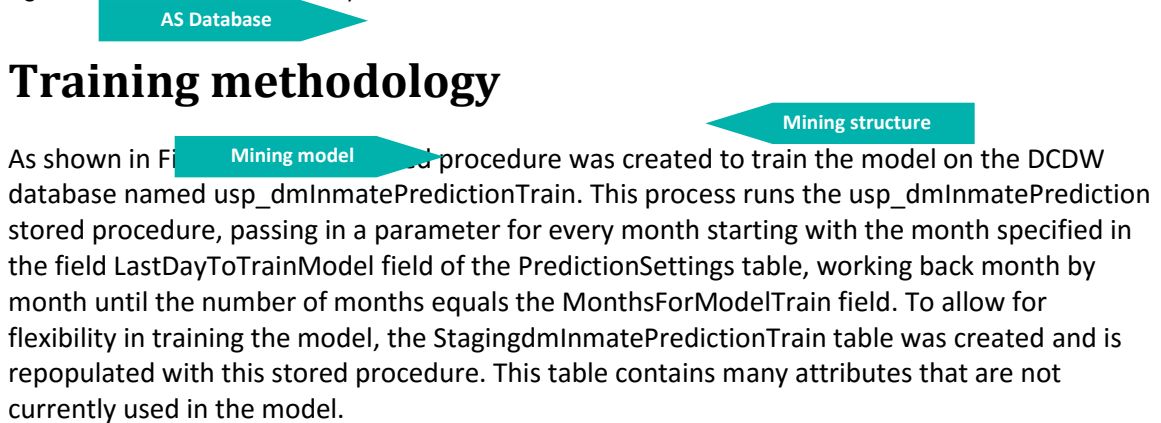
- Dundas Gauge for Reporting Services v1.5
- Dundas Calendar for Reporting Services v1.6
- Dundas Chart for Reporting Services v2.1
- Dundas Map for Reporting Services v1.0

Data mining

Microsoft data mining enables staff to pinpoint past patterns and concerns and understand how they can be used to predict violent events. The data-mining engine enables personnel to find predictive relationships by allowing the application of algorithms to historical attributes. Ultimately, data mining provides a prediction based on past attributes and events. In version 1.0, various inmate attributes were used to predict the probability of violent events. In later releases additional data mining will be performed against staff and facility attributes.

The technology used was SQL Server Analysis Server 2005 and the algorithm was Microsoft Logistic Regression. Figure 50 shows the data mining structure layout.

Figure 50: Data mining structure layout



Training methodology

As shown in Figure 51, a stored procedure was created to train the model on the DCDW database named usp_dmlnmatePredictionTrain. This process runs the usp_dmlnmatePrediction stored procedure, passing in a parameter for every month starting with the month specified in the field LastDayToTrainModel field of the PredictionSettings table, working back month by month until the number of months equals the MonthsForModelTrain field. To allow for flexibility in training the model, the StagingdmlnmatePredictionTrain table was created and is repopulated with this stored procedure. This table contains many attributes that are not currently used in the model.

Figure 51: Training methodology

Table 8 describes the available attributes in the training methodology.

Table 8: Training methodology attributes	
Attribute	Function
Offender_ID	Used for mining key
PriorViolentEvent	Number of events for inmate in the past number of days indicated by DaysToEvaluate (PredictionSettings Table)
PriorNonViolentEvent	Same as PriorViolentEvent but for nonviolent events
Gender	Gender indicator for inmate
Race	Race indicator for inmate
Time_served	Calculated from AdmissionDate to current date
BedMission	BedMission from inmate's bed
Bed_Category	Categorized beds based on bed mission (See Table 9.)

Table 8: Training methodology attributes

Attribute	Function
Event_Count	The actual prediction (In training, however, we supply the data-mining engine this information.)
posDrugTest	A one (1) or zero (0) indicating whether inmate has ever had a positive drug test or refused a drug test (This attribute looks at all of the history and does not use PredictionSettings table.)
Category	The facility category
GangMember	A one (1) or zero (0) indicating whether inmate has ever been a member of a gang or suspected (This attribute looks at all of the history and does not use PredictionSettings table.)
PriorViolentInvestigation	Same as PriorViolentEvent, but filtered for investigations only
PriorUOF	Same as PriorViolentEvent, but filtered for Use of Force (UOF) only
Facility_ID	For reference only (not used in model)
Region	For reference only (not used in model)
BKOBISFacility	For reference only (not used in model)
HighRiskOffender	Inmate indicator
ViolentOffenderFlag	ViolentOffenderFlag for inmate
MonthsSinceLastViolentEvent	The number of months back from the event date or predicted date of last violent event

Table 9 lists the bed categories and bed missions.

Table 9: Bed mission attributes

Bed category	Bed missions
SH	ADMINISTRATIVE CONF
	CLOSE MANAGEMENT I
	CLOSE MANAGEMENT II
	CLOSE MANAGEMENT III
	DISCIPLINARY CONF
	DEATH ROW
	MAXIMUM MANAGEMENT
	PROTECTIVE MGT
GP	ELDERLY AGE 50+
	EMERGENCY BEDS
	ELDERLY AGE 59+
	FIRE STATION

Table 9: Bed mission attributes	
Bed category	Bed missions
	GENERAL POPULATION
	RECEPTION/ORIENTATN
MD	DIABETIC
	HOSPITAL
	INFIRMARY
	INTENSIVE MED. CARE
	ISOLATION-MEDICAL
	OUTSIDE HOSPITAL
	PALLIATIVE CARE
MH	CMHI UNIT BEDS
	CMHI/BEDS
	CSU
	CSU/TCU - DUAL USAGE
	DUAL DIAGNOSIS
	SUICIDE OBSERVATION
	TCU
CN	CHILD NUTRITION
	RES FAITH/CHILD NTRI
	TIER/CHILD NUTRITION
FC	FAITH & CHARACTER
	SELF IMPROVEMENT
YA	YOUNG ADULT OFFENDRS
	YOUTHFUL OFFENDER
PG	DRUG TREATMENT
	SUB. ABUSE TRAN REEN
	TRANSITION/RE-ENTRY

Predictions

As shown in Figure 52, Inmate Predictions are recalculated nightly by the Process DCIS Daily Updates AND Populate DCDW job, Step 4, which in turn runs the SSIS package \DataMining\DataMiningInmates. This package queries the Analysis Server Data-mining engine using the dataset generated by usp_dmlInmatePrediction. This time no parameters are passed so the process uses the current date to base predictions. The output from the data-mining engine is sequentially stored in the Dim_InmatePredictions table. To ease the large storage requirements that would be needed to store daily prediction risks, this job runs nightly; however, it inserts or updates the data as the first of the current month. This will provide for storage of historical information while still minimizing table size.

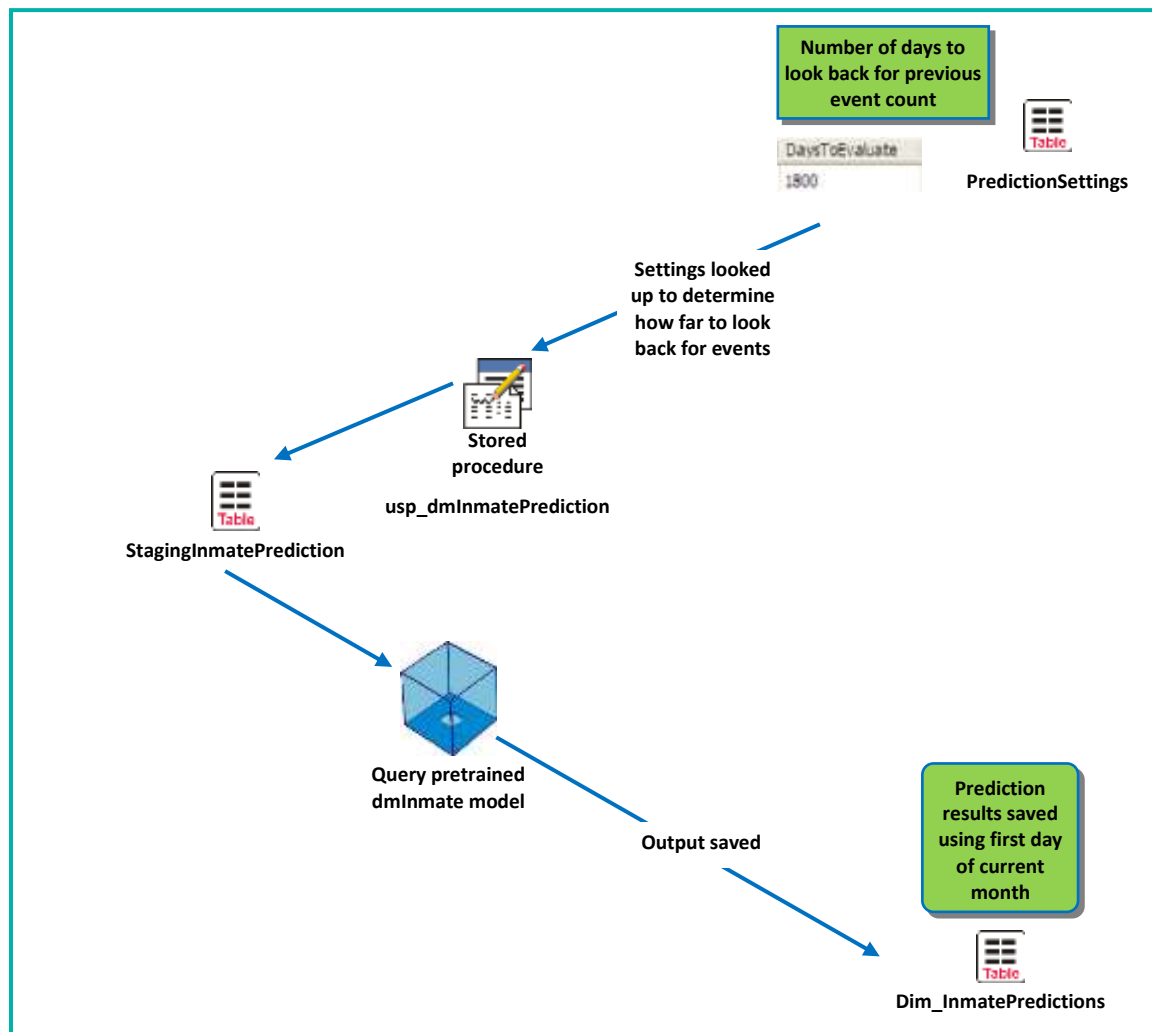


Figure 52: Training methodology

Accuracy

Accuracy is one of the most important things you should consider when building a data-mining application. Having said this, predicting of the future is not an exact science. It is very likely that the inmate model will go through several different versions and may have a need to be re-trained on an ongoing basis.

For the first iteration of training, Idea used the data ending on December 12, 2007 and twelve months earlier. We also used 1800 days to evaluate past events. As shown in figures 53 through 56, Idea used supporting data (lift charts and legends) for the period of January through April, 2008. With multiple iterations of training, these dates were determined to be a good timeframe for predictions as a predetermined holdout of data.

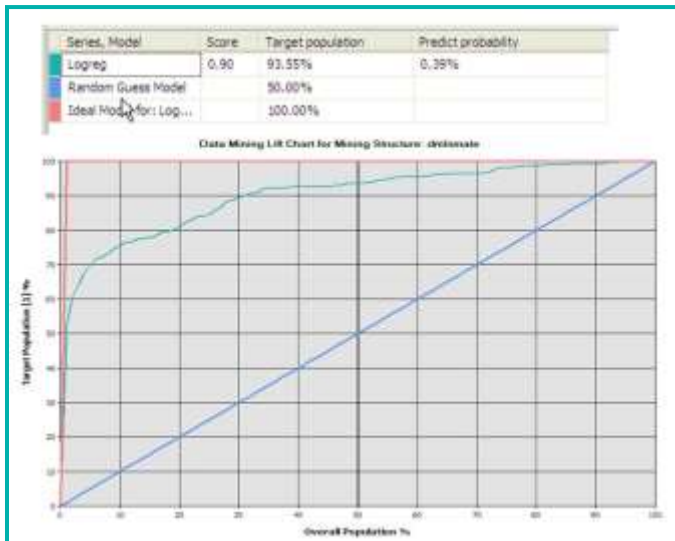


Figure 53: Date of prediction 1/1/2008

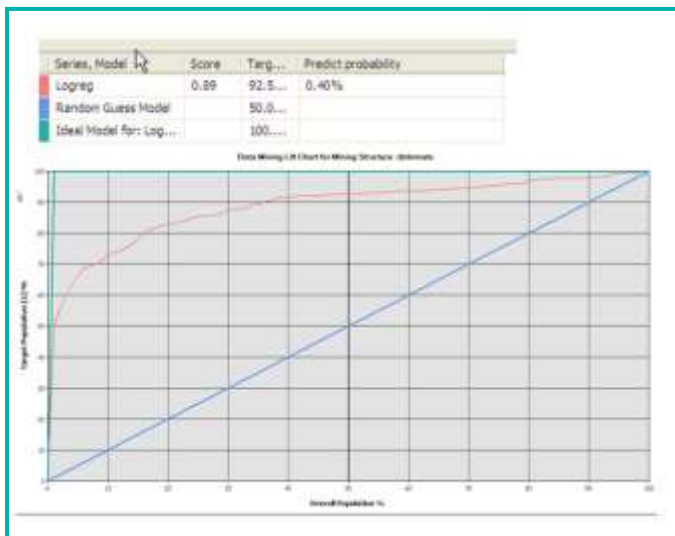


Figure 54: Date of prediction 2/1/2008

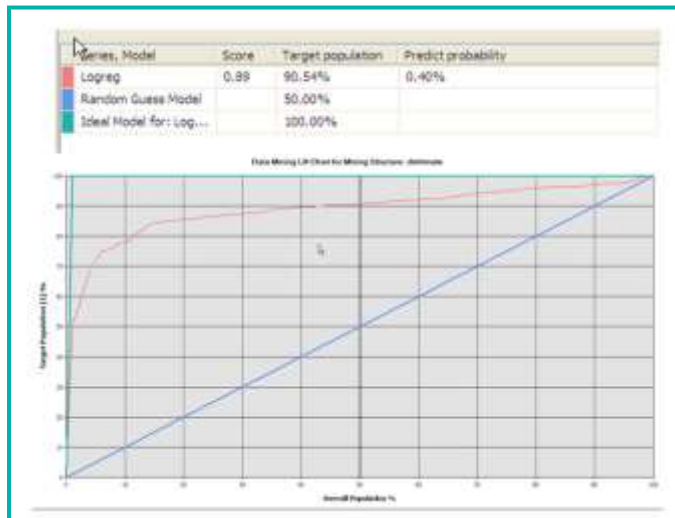


Figure 55: Date of prediction 3/1/2008

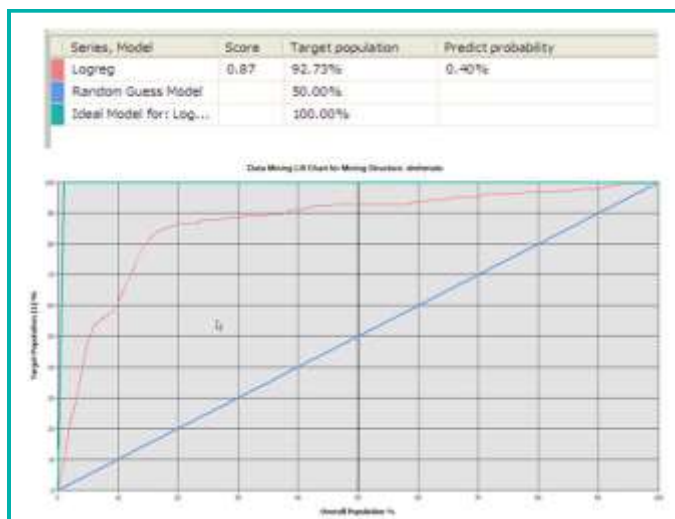


Figure 56: Date of prediction 4/1/2008

Further tests were then completed using most possible combinations of the various inputs available for training of inmates. For easy comparison of different resulting lift charts the Microsoft score was used. This is defined as score and is the (geometric) mean score of all the points constituting the scatter plot. The best scores were found using the following attributes. These scores were all between .87 and .90. As shown in Figure 57, the attributes marked “ignore” were found to have no effect or a negative effect on the score value.

Structure	Logreg
	Microsoft_Logistic_Regression
Bed Category	Input
Bed Mission	Ignore
BKOBIS Facility	Ignore
Category	Ignore
Current Month	Ignore
Event Count	PredictOnly
Facility ID	Ignore
Gang Member	Ignore
Gender	Input
High Risk Offender	Ignore
Months Since Last Violent Ev...	Input
Offender ID	Key
Pos Drug Test	Input
Prediction Date	Ignore
Prior Non Violent Event	Input
Prior UOF	Ignore
Prior Violent Event	Input
Prior Violent Investigation	Ignore
Race	Input
Region	Ignore
Time Served	Input
Violent Offender Flag	Input
XCM	Input

Figure 57: Currently used variables for training

Personnel

Analyst

Analyzing predictive data requires specialized skills. In addition to basic educational and technical qualifications, an analyst should at a minimum possess a statistics background and three years of experience with the use of predictive analytics tools such as SAS (Microsoft preferred).

Database administrator

The administration of COTAS databases is the same as it would be for any other database. In addition to basic educational and technical qualifications, a database administrator should possess two years of OLAP administration experience.

Developer/programmer

Data mining is a very specialized area. In addition to basic educational and technical qualifications, a developer should possess the following minimum qualifications and experience: three years of experience with version 2005 or greater, seven years of experience with general SQL Server development, three years of OLAP experience and two years of data mining experience.

Glossary

Department terms

COTAS	Correctional Operations Trend Analysis System
Active inmate	Inmates that are currently part of the inmate population in a facility
Inactive inmate	Former inmates that are no longer part of the inmate population, but could be monitored by parole
Offender	A combination of inactive and active inmates (In DCIS an inmate can be an offender, but an offender is not always an inmate.)
Events	Violent and nonviolent occurrences that are recorded at the facility and central office level Violent events ▪ Disciplinary reports ▪ Escapes ▪ Investigations ▪ Use of Force (UOF) Nonviolent events ▪ Central Office grievances ▪ Disciplinary reports ▪ Escapes ▪ Field grievances ▪ Investigations
Predictors	A function that produces a risk score of a violent event based on inmate attributes and prior history of events
Facility categories	Eight facility categories are used to define how facilities are compared to one another: ▪ Work Release Center ▪ Work Camps and Jails ▪ Work Camps ▪ Institutions, Level 4 ▪ Institutions, Level 5 ▪ Close Management Institutions ▪ Female Institutions ▪ Reception Centers

OBIS	Offender Based Information System
DCIS	A relational database used to build a data warehouse
IGLOGS	Inspector General Logs: the database used to load field grievances, central office grievances, investigations and use of force (This database is a source of external data and contains the Inspector General Logs on investigations and use of force. Its schema is provided because an ETL process points to it.)
RMS	Roster Management System: The database used to load staff information (This database is a source of external data and contains the Roster Management System data on security staff and their scheduling. Its schema is provided because an ETL process points to it.)
DCDW	Department of Corrections Data Warehouse: the data warehouse used for COTAS

Technical terms

Data mining	The act of sorting through data to identify patterns and establish relationships to predict future events
Dashboard	A user interface that organizes and presents information in a way that is easy to read.
Cube	The main objects in online analytic processing (OLAP), a technology that provides fast access to data in a data warehouse (A cube is a set of data that is usually constructed from a subset of a data warehouse and is organized and summarized into a multidimensional structure defined by a set of dimensions and measures.)
Data warehouse	A central repository for all or significant parts of the data that an enterprise's various business systems collect.
SSIS	SQL Server Integration Service is the data transformation standard for SQL Server 2005
SSAS	SQL Server Analysis Service is the developing tool for creating cubes, KPIs and working with data mining
SSRS	SQL Server Reporting Service is the developing tool for creating reports

Appendix

Appendix A - Packages Developed

Audit PreExecute of Package

The Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. As shown in Figure 58, it contains seven input variables, one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

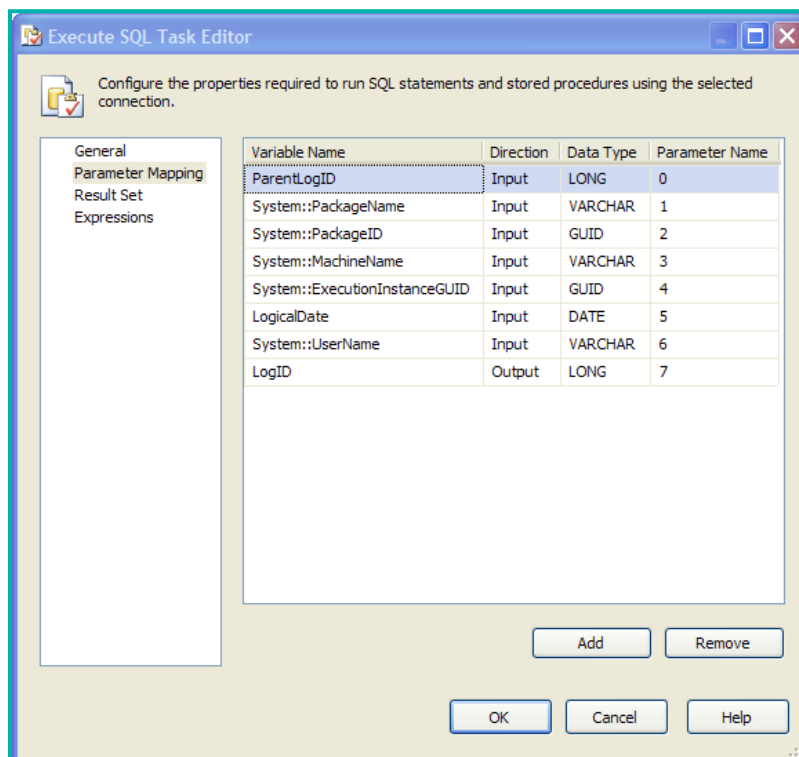


Figure 58: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Create Temp Table #Updates

By running the following execute SQL statement, Create Temp Table #Updates creates a temporary table on the REDACTED server to hold offender data rows that have changed:

```
CREATE TABLE [#Updates]
(
    [Facility_ID] INTEGER
)
```

Set EndTime Variable

By running the following execute SQL statement, Set EndTime Variable sets the EndTime variable that later helps return only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_Facility Inserts task

As shown in Figure 59, the Dim_Facility Inserts data flow task uses the usp_Dim_Facility stored procedure from the DCIS database on the REDACTED server to update the Dim_Facility table in the data warehouse.

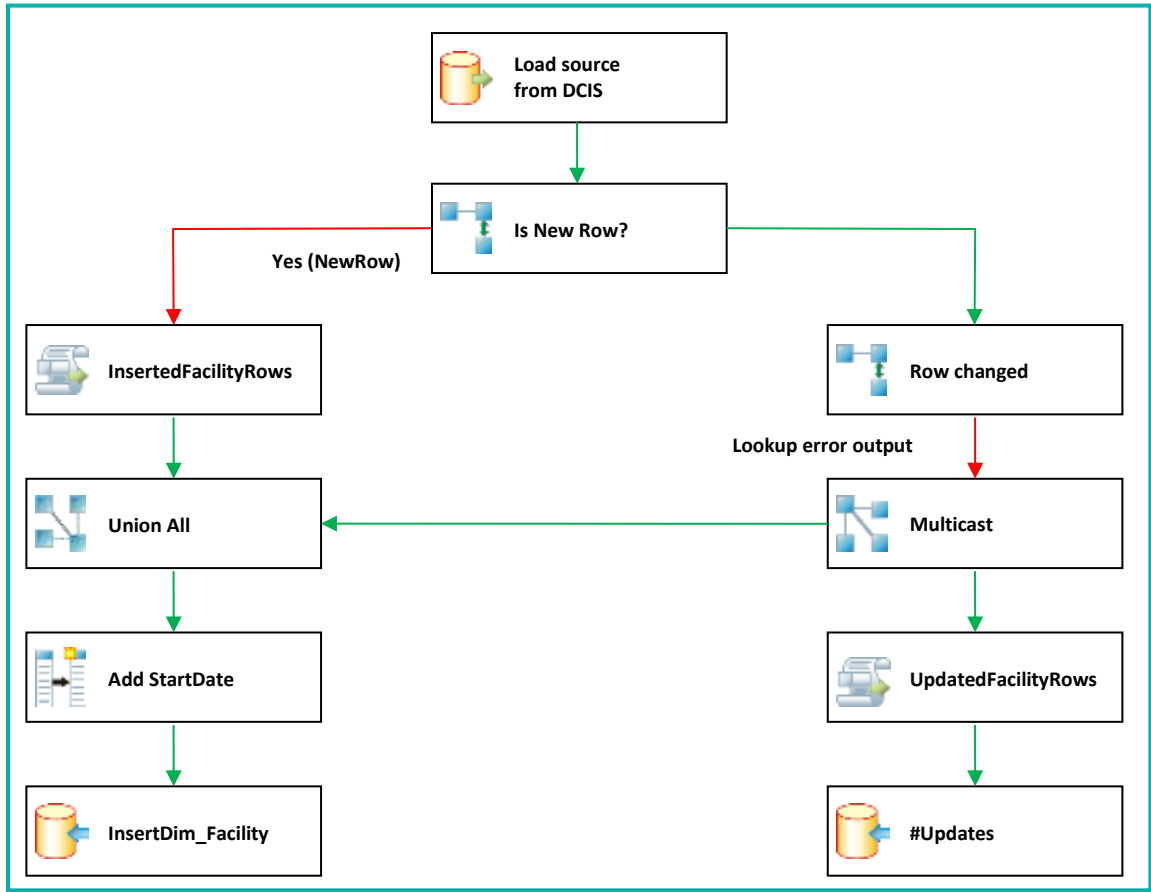


Figure 59: Dim_Facility Inserts data flow task

Table 10 describes the components of the Dim_Facility Inserts data flow task.

Table 10: Components of the Dim_Facility Inserts task		
Symbol	Data flow component	Description
	Load source from DCIS	Captures columns from the stored procedure usp_Dim_Facility from source database DCIS

Table 10: Components of the Dim_Facility Inserts task

Symbol	Data flow component	Description
	Is New Row?	Lookup that checks to see if any new rows are being loaded: <pre>SELECT BKOBISFacility, Facility_ID FROM DIM_Facility WHERE (RowEndDate IS NULL)</pre>
	InsertedFacilityRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Row Changed	Lookup that checks to see if the row is not new has it been modified: <pre>SELECT Facility_ID, BKOBISFacility, FacilityCodeKey, Facility, Category, FacilityType, AddressLine1, AddressLine2, City, Zip, Region, ServiceCenter, FacilityHeadName, FacilityHeadTitle, PhoneNumber, SuncomNumber, FaxNumber, InsertAuditID, UpdateAuditID, RowStartDate, RowEndDate, County FROM DIM_Facility</pre>
	Multicast	Sends the results of the lookup to #Updates and Dim_Facility tables
	UpdatedFacilityRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	#Updates	Places updated rows in #Updates temp table
	Union All	Joins newly entered facility rows and old rows that have been modified
	Add StartDate	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database
	Insert Dim_Facility	Brings together needed columns to be placed in Dim_Facility table

Dim_Facility Mark deleted task

As shown in Figure 60, the Dim_Facility Mark deleted data flow task uses the newly created Dim_Facility table from the DCDW database on the REDACTED server to load the #Updates temp table.

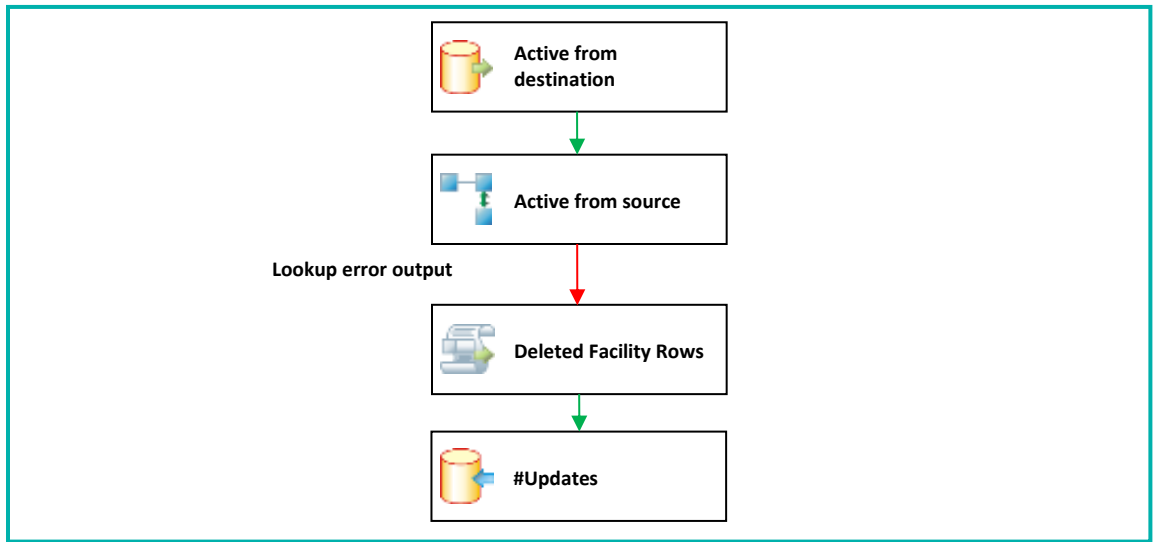


Figure 60: Dim_Facility Mark deleted data flow task

Table 11 describes the components of the Dim_Facility Mark deleted data flow task.

Table 11: Components of the Dim_Facility Mark deleted task		
Symbol	Data flow component	Description
	Active from Destination	Captures BKOBISFacility and Facility_ID columns from the table just created in the last data flow DIM_Facility from source database DCDW: SELECT BKOBISFacility, Facility_ID FROM DIM_Facility WHERE (RowEndDate IS NULL)
	Active from Source	Lookup that captures BKOBISFacility to be placed in the temp table #Updates
	Deleted Facility Rows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	#Updates	Brings together needed columns to be placed in #Updates temp table

Update EndDates

By running the following execute SQL statement, Update EndDates execute SQL task refreshes the RowEndDate column and UpdateAuditID columns with data gathered from both DIM_Facility and #Updates:

```
Update f set RowEndDate=?,UpdateAuditID=?
FROM Dim_Facility f
JOIN #Updates u on u.facility_ID=f.facility_ID
```

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime with the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 12 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 12: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceDCIS		REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 Destination Temp		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationUpdates.dtsConfig

Dim_DormBed

As shown in Figure 61, the Dim_DormBed package updates the table Dim_Dormbed by using the stored procedure usp_Dim_DormBed to load the DCDW data warehouse on the REDACTEDserver. In addition, this stored procedure extracts all the dorm beds in all the facilities in Florida gathered from the server REDACTEDn the DCIS database. Finally, it adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTEDserver.

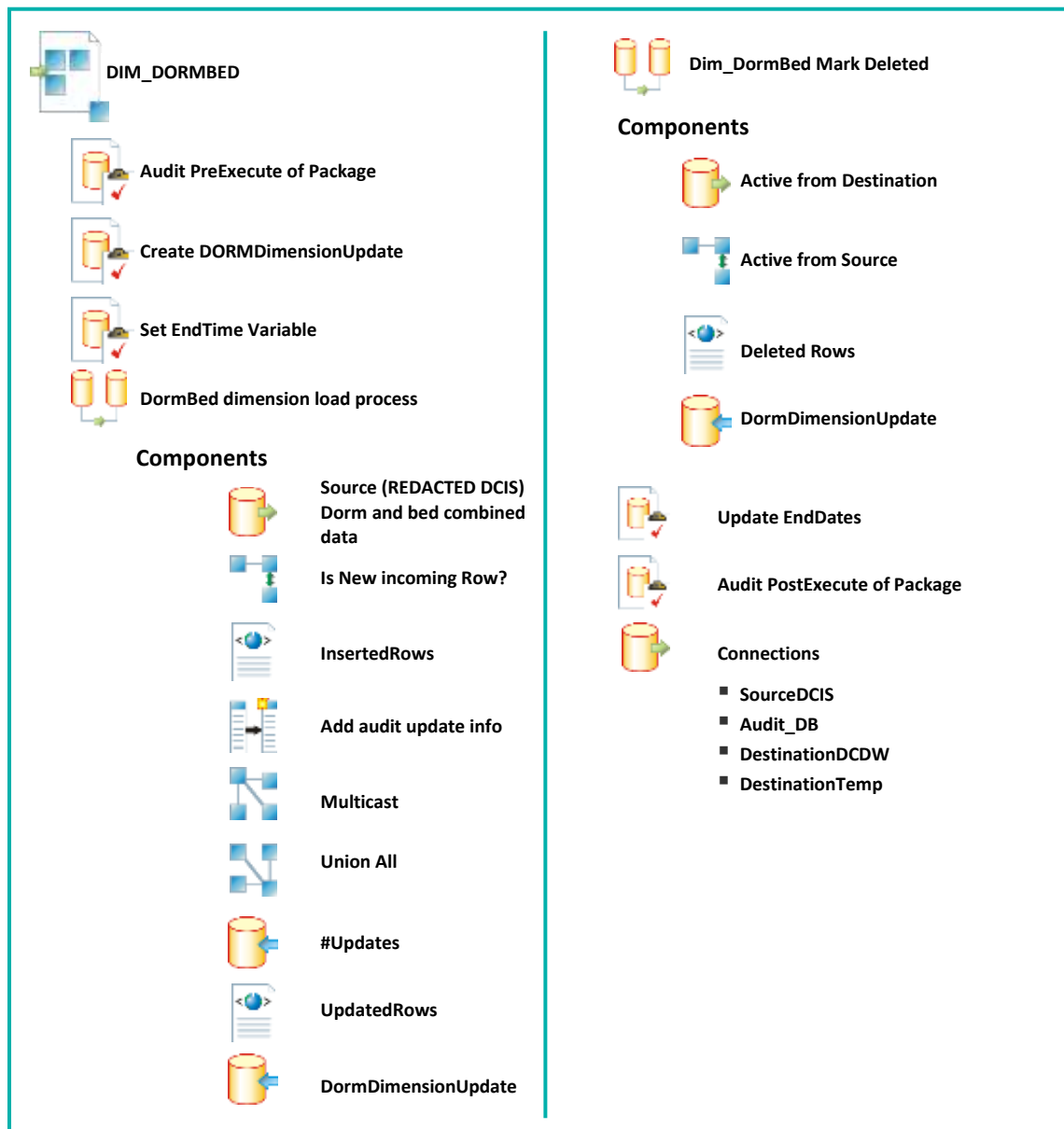


Figure 61: Dim_DormBed package

Audit PreExecute of Package

As shown in Figure 62, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

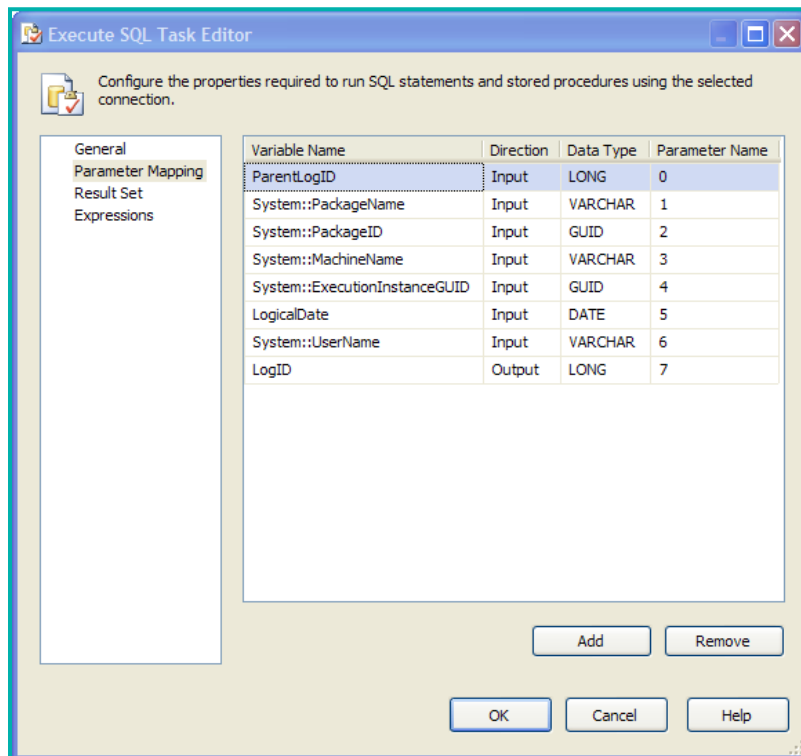


Figure 62: Execute SQL Task Editor dialog box for the Audit PreExecute of Package execute SQL task

Create DormDimensionUpdate

By running the following execute SQL statement, Create DormDimensionUpdate creates a temporary table on the REDACTEDserver to hold offender data rows that have changed:

```
CREATE TABLE [#Updates]
    [BKOBISFacility] varchar(4),
    [BKOBISBed] varchar(6)
)
```

Set EndTime Variable

By running the following execute SQL statement, Set EndTime Variable the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

DormBed dimension load process

As shown in Figure 63, the DormBed dimension load process data flow task uses the usp_Dim_DormBed stored procedure from the DCIS database on the REDACTEDserver to update the Dim_DORMBED table in the data warehouse.

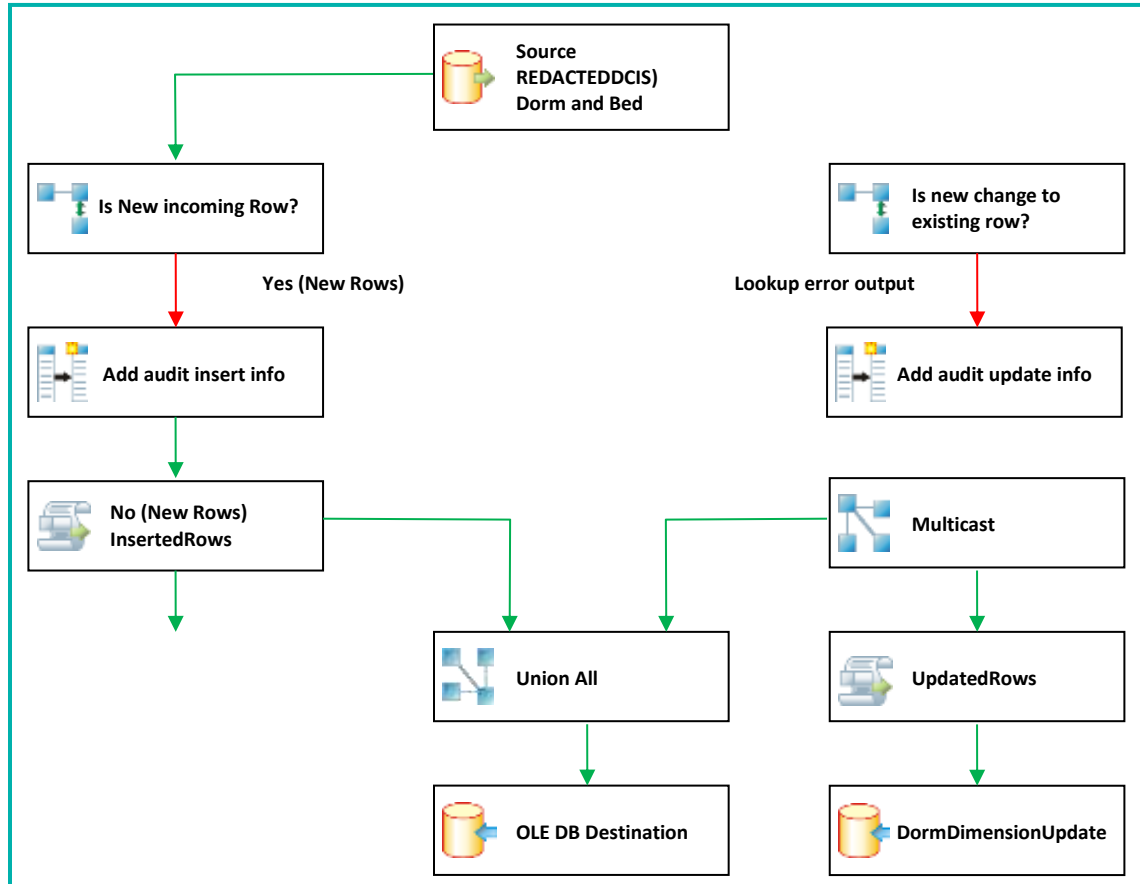


Figure 63: DormBed dimension load process data flow task

Table 13 describes the components of the DormBed dimension load process data flow task.

Table 13: Components of the DormBed dimension load process task		
Symbol	Data flow component	Description
	Source (REDACTEDDCIS) Dorm and Bed combined data	Captures columns from the stored procedure usp_Dim_DormBed from source database DCIS
	Is New incoming Row?	Lookup that checks to see if any new rows are being loaded: <pre> SELECT BKOBISFacility, BKOBISBed FROM dbo.DIM_DORMBED WHERE (RowEndDate IS NULL) </pre>
	Add audit update info	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database

Table 13: Components of the DormBed dimension load process task

Symbol	Data flow component	Description
	No (NewRows) InsertedRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Is New change to existing Row?	Finds out if the changes that have been made are to rows that existed prior to load: <pre>SELECT * FROM dbo.DIM_DORMBED WHERE RowEndDate IS NULL</pre>
	Add StartDate	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database
	Multicast	Sends the results of the lookup to #Updates and Dim_DORMBED tables
	Union All	Joins newly entered DORMBED rows and old rows that have been modified
	UpdatedFacilityRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	OLE DB Destination	Brings together needed columns to be placed in DORMBED table
	UpdatedRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	DormDimensionUpdate	Places updated rows in #Updates temp table

Dim_DormBed Mark Deleted

As shown in Figure 64, the Dim_DormBed Mark Deleted data flow task uses the newly created Dim_ DORMBED table from the DCDW database on the REDACTEDserver to load the #Updates temp table.

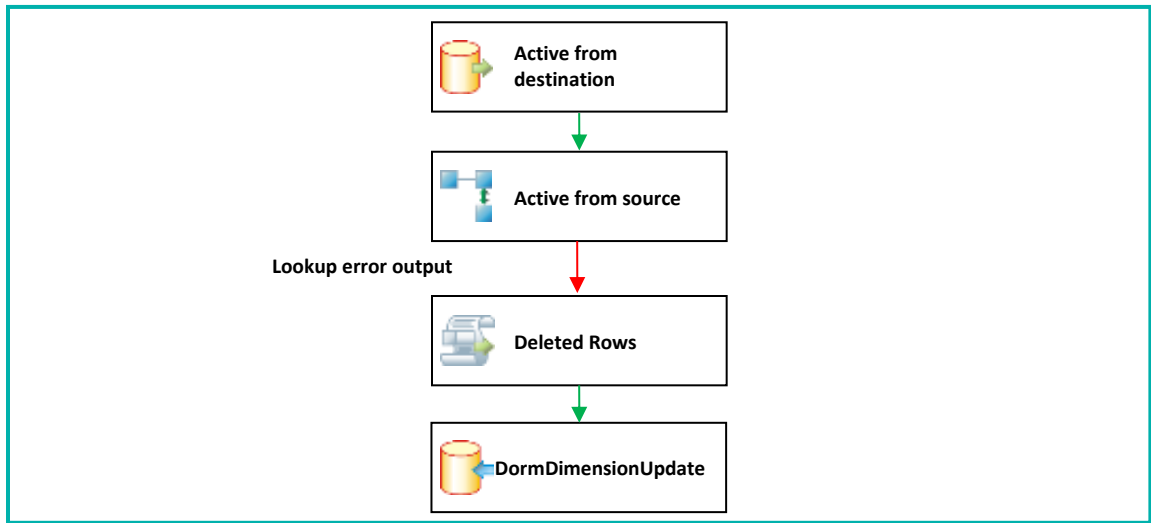


Figure 64: Dim_DormBed Mark Deleted data flow task

Table 14 describes the components of the DormBed Mark Deleted data flow task.

Table 14: Components of the Dim_DormBed Mark Deleted data flow task		
Symbol	Data flow component	Description
	Active from Destination	Captures BKOBISFacility and Facility_ID columns from the table just created in the last data flow DIM_Facility from source database DCDW: <code>SELECT BKOBISFacility, BKOBISBed</code> <code>FROM DIM_DORMBED</code> <code>WHERE (RowEndDate IS NULL)</code>
	Active from Source	Lookup that captures BKOBISFacility to be placed in the temp table #Updates <code>SELECT BKOBISFacility, BKOBISBed</code> <code>FROM V_DIM_DORMBED</code>
	Deleted Facility Rows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	DormDimensionUpdate	Brings together needed columns to be placed in #Updates temp table

Update EndDates

By running the following execute SQL statement, Update EndDates refreshes the RowEndDate column and UpdateAuditID columns with the latest data from DIM_DORMBED and #Updates:

```
Update DIM_DORMBED set RowEndDate=?,UpdateAudit_ID=?
FROM DIM_DORMBED
JOIN #Updates u on u.BKOBISFacility= DIM_DORMBED.BKOBISFacility
AND u.BKOBISBed= DIM_DORMBED.BKOBISBed
```

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 15 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 15: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceDCIS		REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 Destination Temp		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationUpdates.dtsConfig

Dim_Gang

The Dim_Gang package uses stored procedures to update tables and load the DCDW data warehouse on the REDACTEDserver. The stored procedures include usp_DWGang_Details, usp_Dim_Gang, usp_DWGang_Allies and usp_DWGang_Enemy, which update the following tables:

- Dim_Gang
- Dim_Gang_Allies
- Dim_Gang_Enemies
- Dim_Gang_Offender

As shown in Figure 65, these stored procedures and related tasks extract all the gang data gathered from the servers REDACTED in the STG and DCIS databases. Gang data includes the gang allies, enemies and the offenders that have gang affiliations. It also adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTED server.

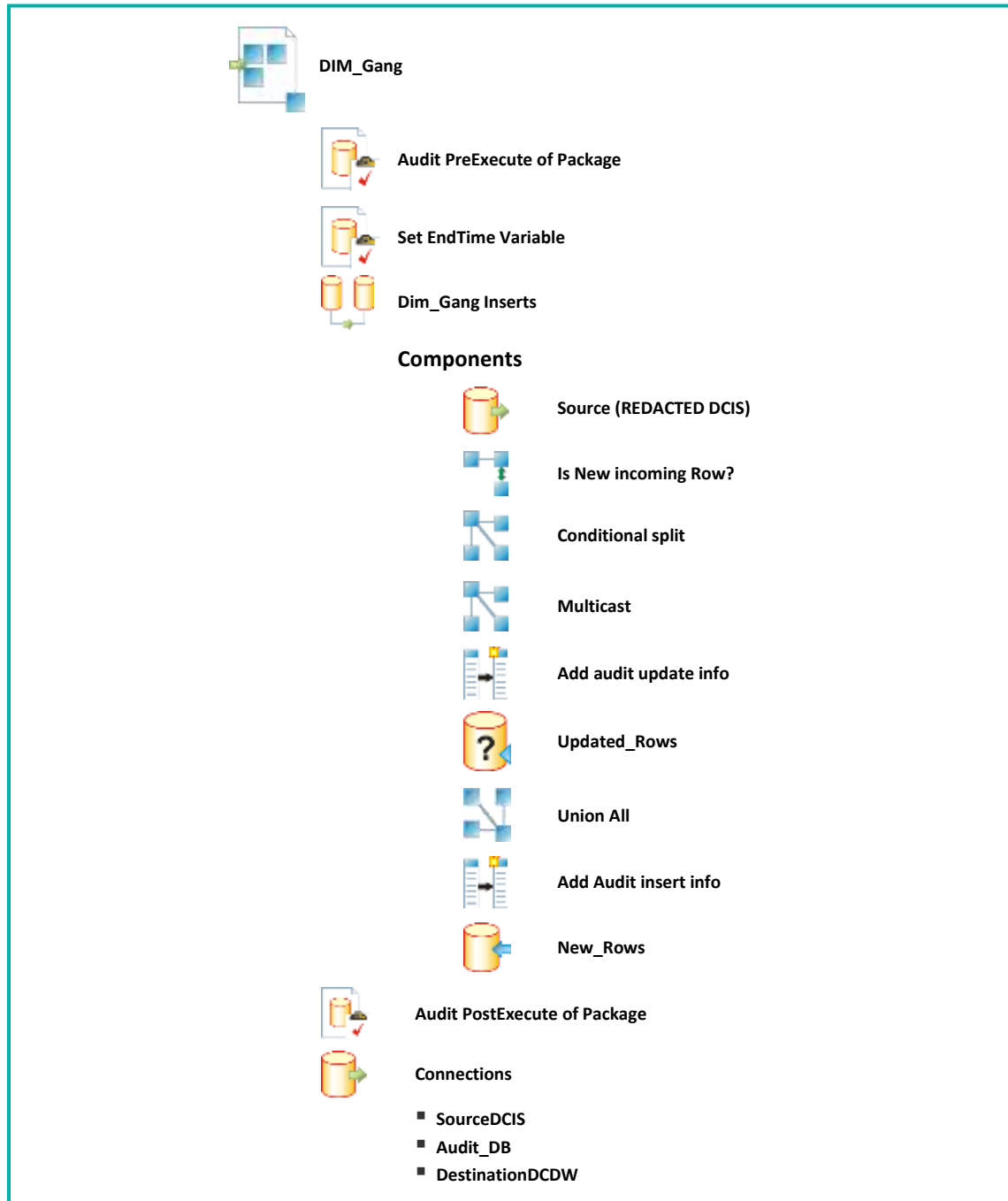


Figure 65: Contents of the Dim_Gang package

Audit PreExecute of Package

As shown in Figure 66, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

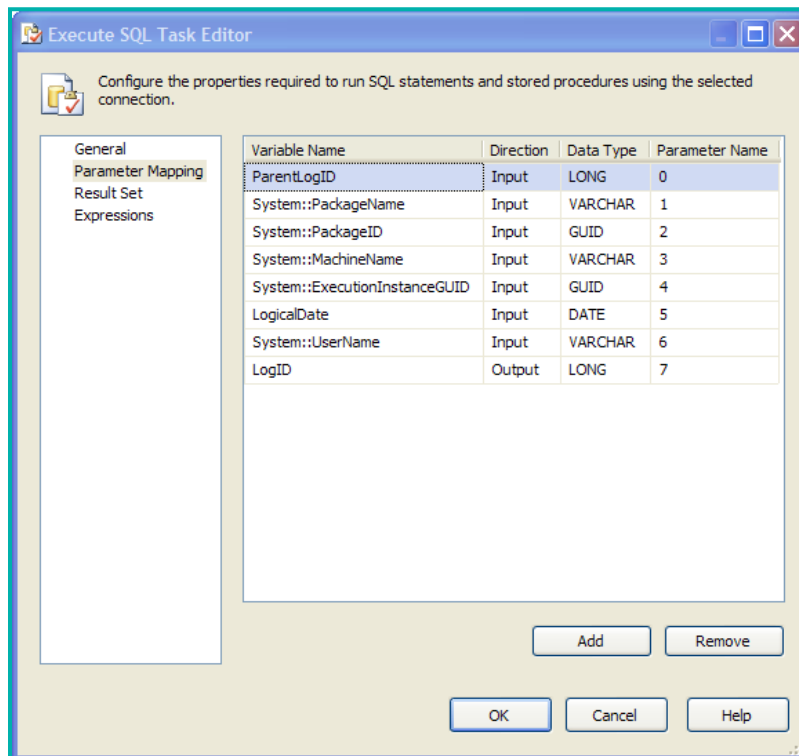


Figure 66: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statements, the Set EndTime Variable execute SQL task sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_Gang Inserts

As shown in Figure 67, this data flow task uses the usp_Dim_Gang stored procedure from the DCIS database on the REDACTED server to update the Dim_Gang table in the data warehouse.

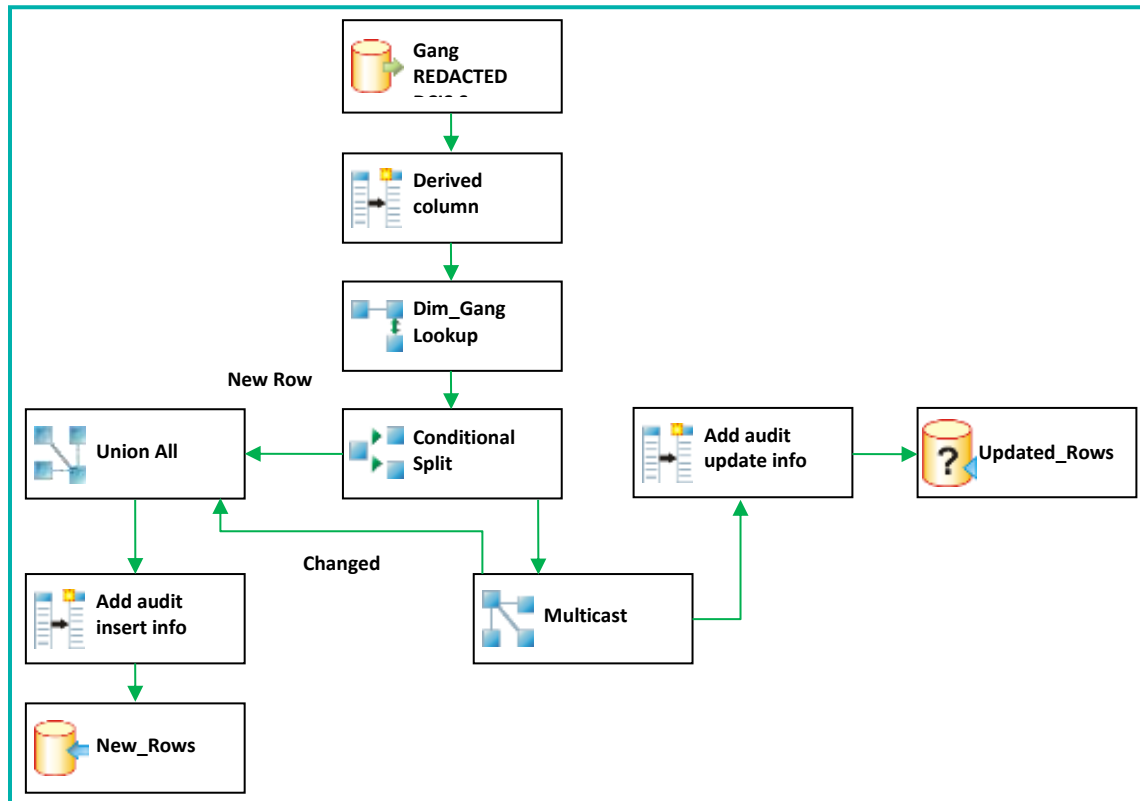


Figure 67: Gang dimension load-process load process task

Table 16 describes the components of the Dim_Gang Inserts load process task.

Table 16: Components of the Dim_Gang Inserts load process task		
Symbol	Data flow component	Description
	Gang REDACTEDDCIS source	Captures columns from the stored procedure usp_Dim_Gang from source database DCIS
	Add audit update info	Derived Column transform that adds AuditLogId column in required tables in the Audit_DB database
	Is New change to existing Row?	Lookup that checks to see if any new rows are being loaded
	Derived Column	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database
	Conditional Split	Conditional split that tests to see if rows are new or changed

Table 16: Components of the Dim_Gang Inserts load process task

Symbol	Data flow component	Description
	Multicast	Sends the results of the conditional split to Updated Rows and New Rows destinations
	Add Audit Update Info	Derived column transform that adds UpdateAudit_ID and RowEndDate columns
	Updated_Rows	UpdateAudit_ID and RowEndDate are set in table
	Union All	Joins newly entered gang rows and old rows that have been modified
	Add Audit Insert Info	Derived column transform that adds InsertAudit_ID and RowStartDate columns
	New_Rows	Brings together needed columns to be placed in Dim_Gang table

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 17 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 17: Connections for the Audit PostExecute of Package task

Connection	Target server	Target database	Configuration file
 SourceDCIS	REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 Audit_DB	REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW	REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

Dim_DrugTest

As shown in Figure 68, the Dim_DrugTest package updates the Dim_DrugTest table by using the stored procedure usp_Dim_DrugTest to load the DCDW data warehouse on the REDACTED server. In addition, this stored procedure extracts the entire inmate drug testing data gathered from the server REDACTED in the DCIS database. Finally, it adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTED server.

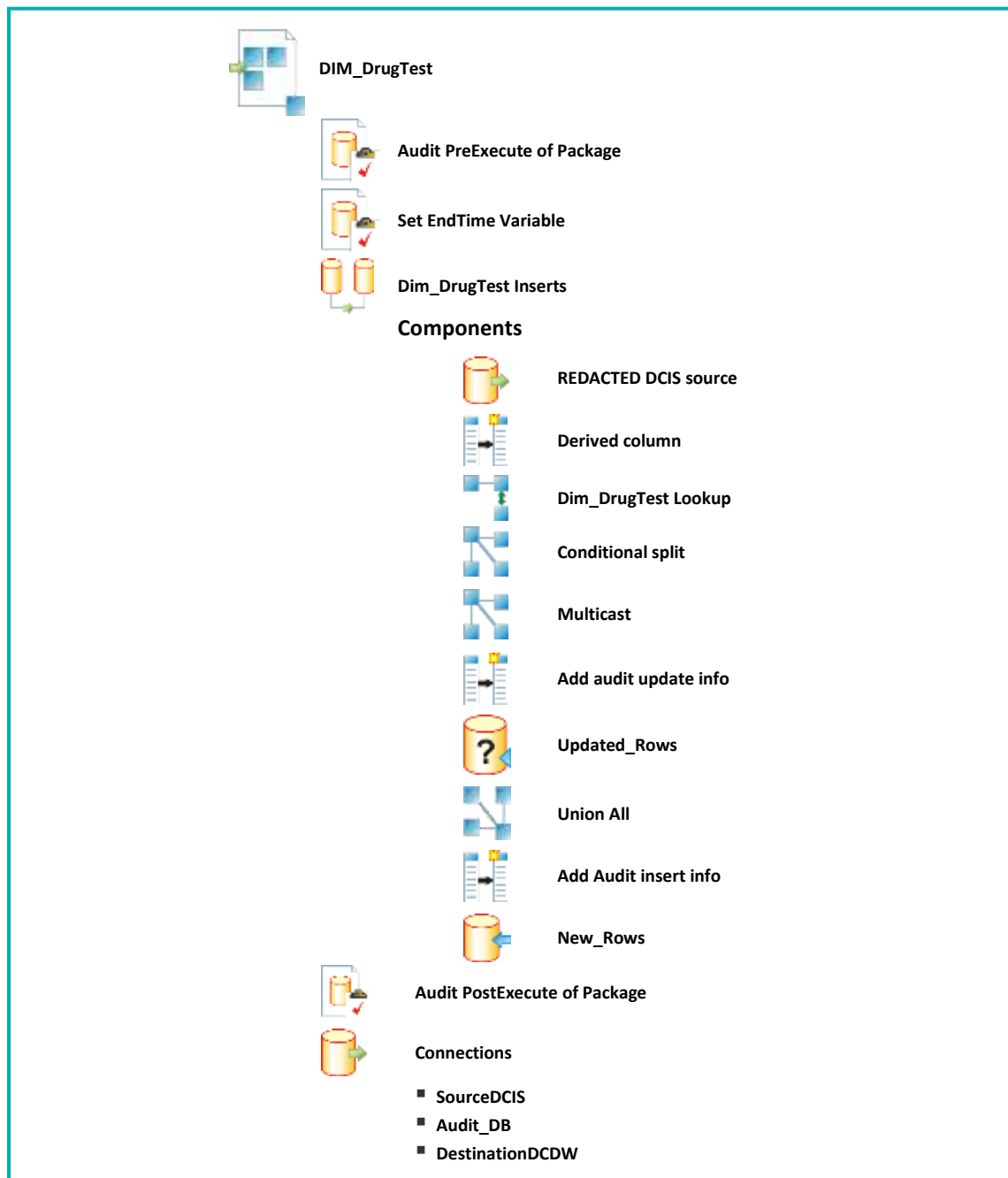


Figure 68: Contents of the Dim_DrugTest package

Audit PreExecute of Package

As shown in Figure 69, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

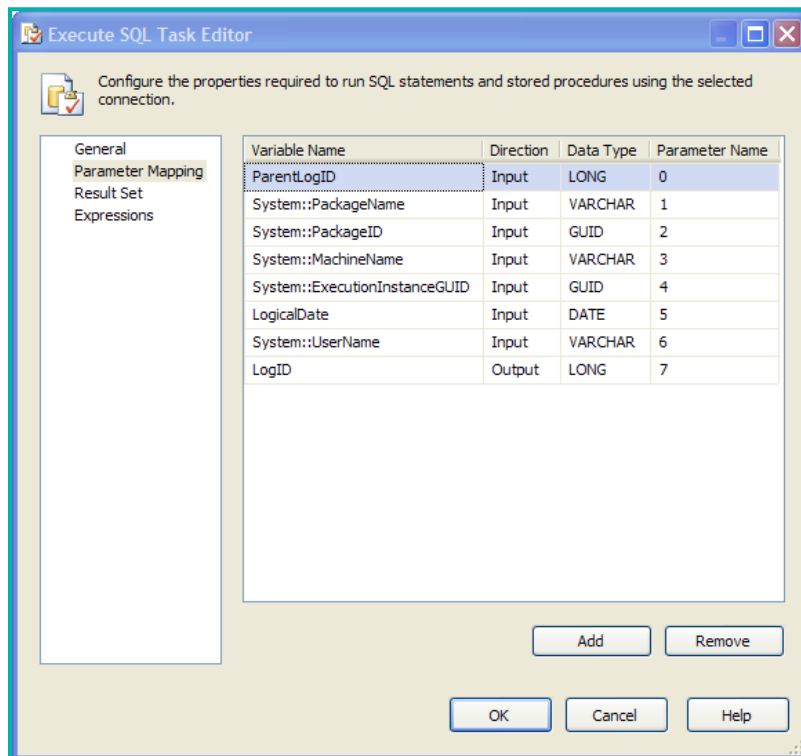


Figure 69: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following execute SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_DrugTest Inserts

As shown in Figure 70, the Dim_DrugTest Inserts data flow task uses the usp_Dim_DrugTest stored procedure from the DCIS database on the REDACTED server to update the Dim_DrugTest table in the data warehouse.

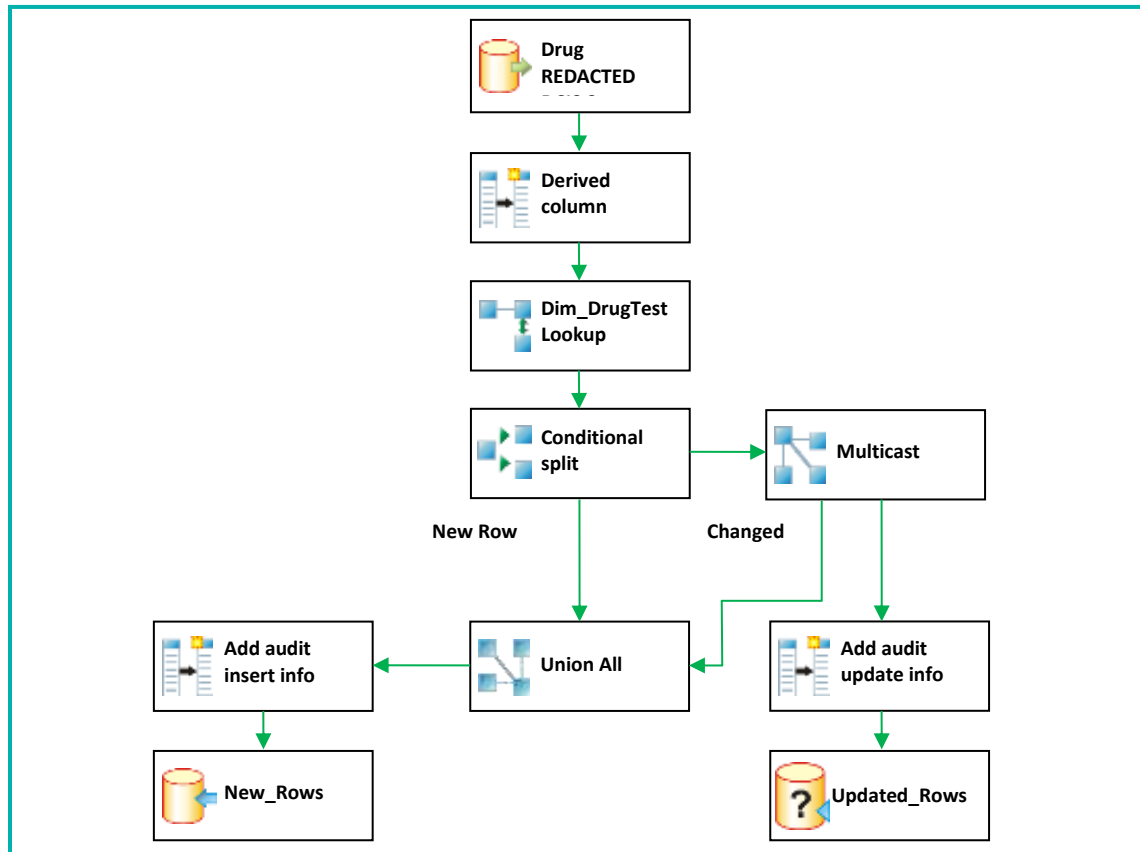


Figure 70: Dim_DrugTest dimension load process data flow task

Table 18 describes the components of the Drug_Test dimension load process task.

Table 18: Components of the Drug_Test dimension load process task		
Symbol	Data flow component	Description
	Drug REDACTED DCIS source	Captures columns from the stored procedure usp_Dim_DrugTest from source database DCIS
	Derived Column	Derived Column transform that adds AuditLogId column in required tables in the Audit_DB database
	Dim_DrugTest Lookup	Lookup that checks to see if any new rows are being loaded
	Conditional Split	Conditional split that tests to see if rows are new or changed

Table 18: Components of the Drug_Test dimension load process task

Symbol	Data flow component	Description
	Multicast	Sends the results of the conditional split to Updated Rows and New Rows destinations
	Add Audit Update Info	Derived column transform that adds UpdateAudit_ID and RowEndDate columns
	Updated_Rows	UpdateAudit_ID and RowEndDate are set in table
	Union All	Joins newly entered gang rows and old rows that have been modified
	Add Audit Insert Info	Derived column transform that adds InsertAudit_ID and RowStartDate columns
	New_Rows	Brings together needed columns to be placed in Dim_DrugTest table

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 19 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 19: Connections for the Audit PostExecute of Package task

Connection	Target server	Target database	Configuration file
 SourceDCIS	REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 Audit_DB	REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW	REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

Dim_Offender

As shown in Figure 71, the Dim_Offender package updates the table Dim_Offender by using the stored procedure usp_Dim_Offender to load the DCDW data warehouse on the REDACTEDserver. In addition, this stored procedure extracts the entire inmate roster with attributes gathered from the server REDACTEDin the DCIS database. Finally, it adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTEDserver.

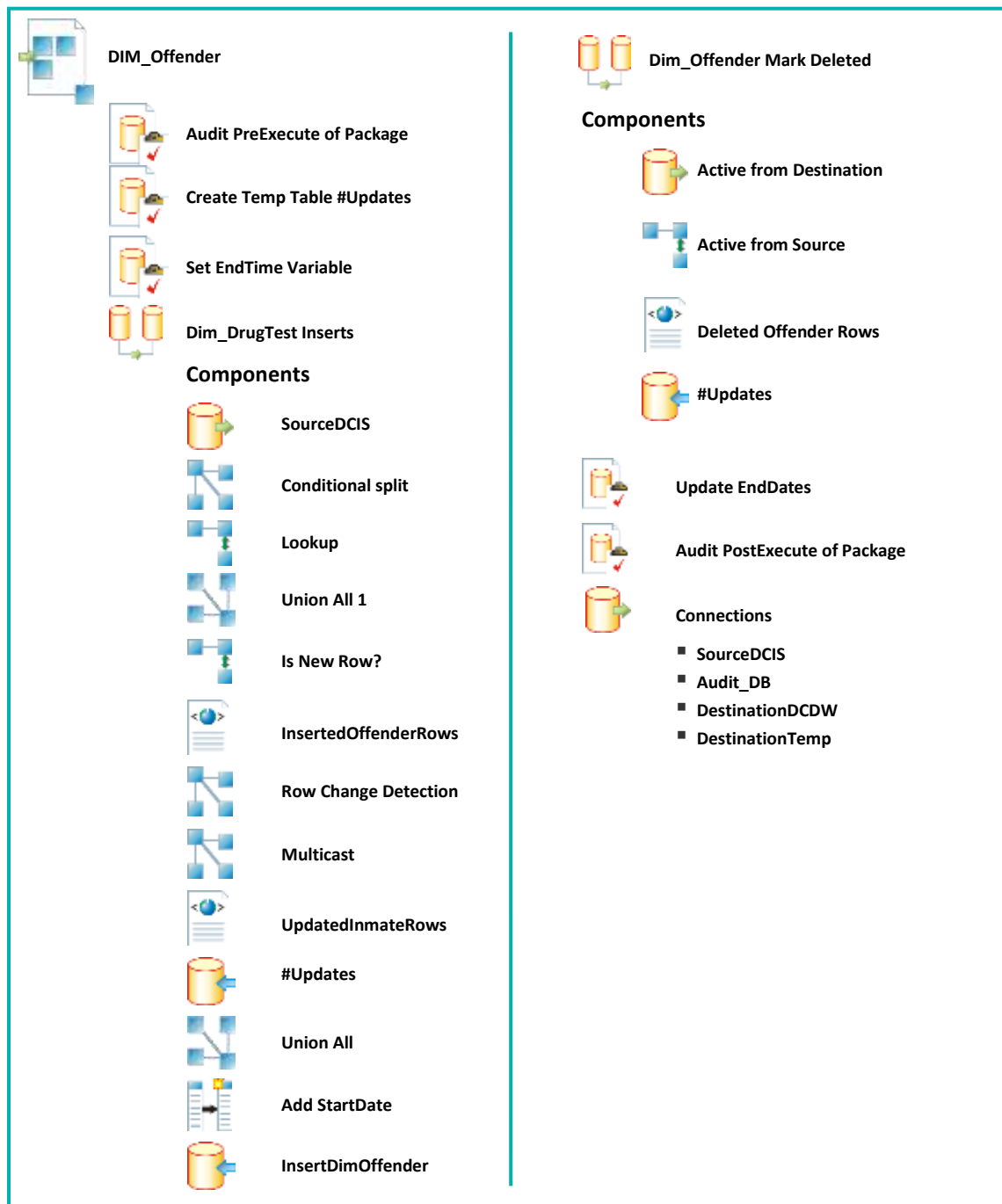


Figure 71: Contents of the Dim_DrugTest package

Audit PreExecute of Package

As shown in Figure 72, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

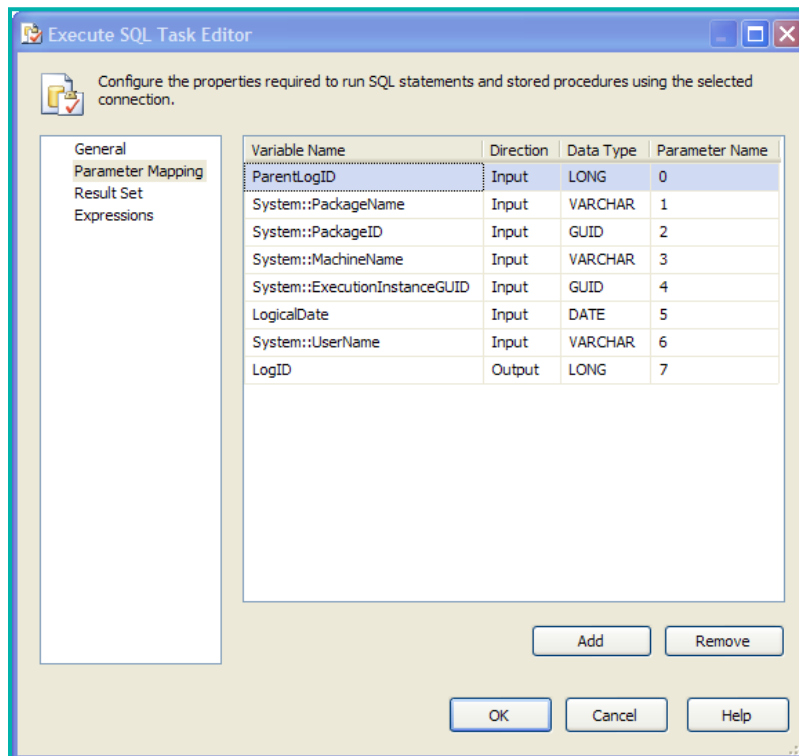


Figure 72: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Create Temp Table #Updates

By running the following execute SQL statement, Create Temp Table #Updates creates a temporary table on the REDACTED server to hold offender data rows that have been changed:

```
CREATE TABLE [#Updates]
(
    [Offender_ID] INTEGER
)
```

Set EndTime Variable

By running the following execute SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_Offender Inserts

As shown in Figure 73, the Dim_Offender Inserts data flow task uses the usp_Dim_Offender stored procedure from the DCIS database on the REDACTED server to create the Dim_Offender table in the data warehouse.

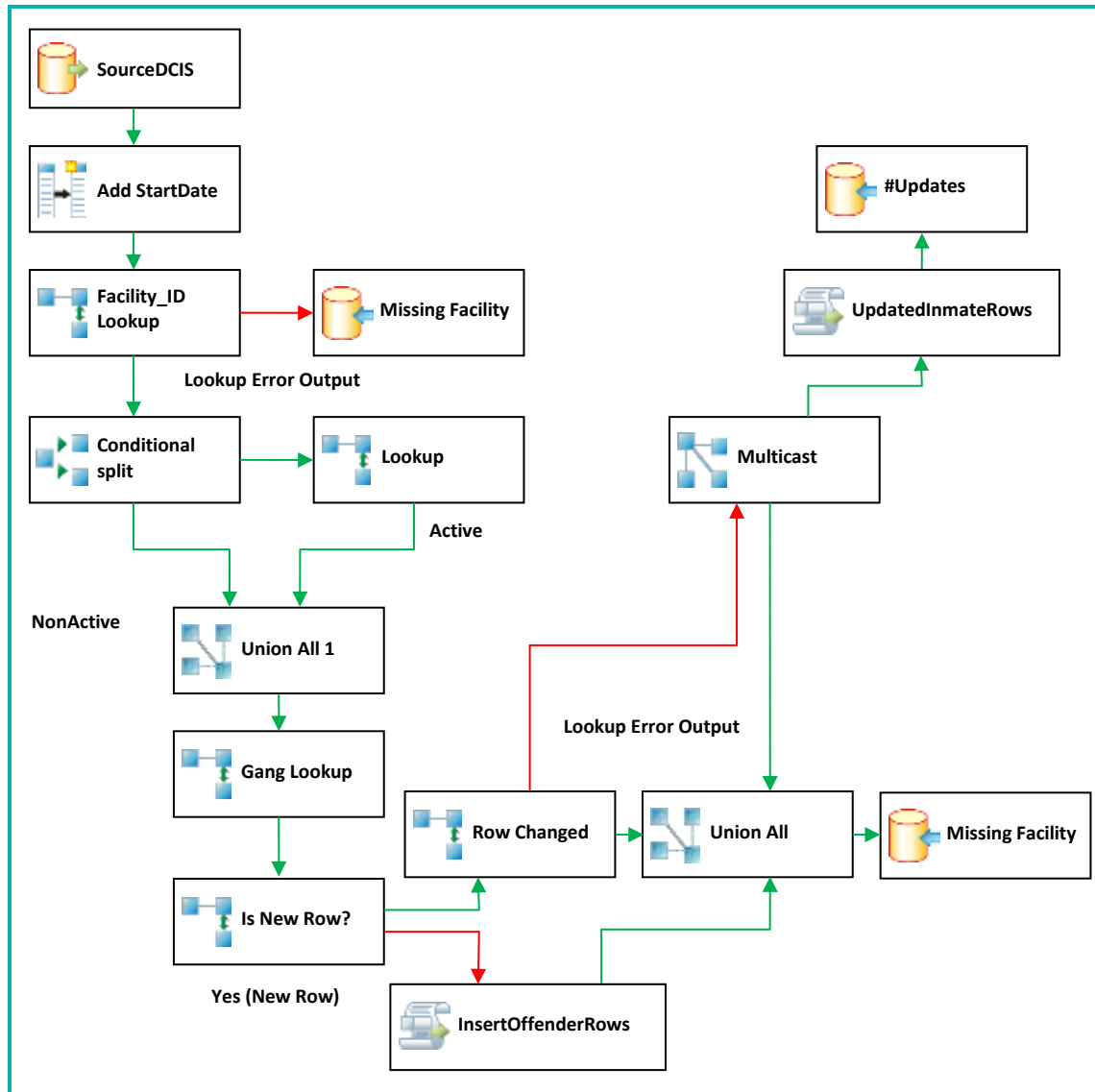


Figure 73: Dim_Offender Inserts load process data flow task

Table 20 describes the components of the Dim_Offender Inserts data flow task.

Table 20: Components of the Dim_Offender load process task		
Symbol	Data flow component	Description
	SourceDCIS	Captures columns from the stored procedure usp_Dim_Offender from source database DCIS
	Add StartDate	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database
	Facility_ID Lookup	Lookup transform that retrieves the Dim_FacilityID by matching source facilities with those already in the data warehouse table Dim_Facility
	Missing Facility	SQL task that places facility IDs with no matching records in the Dim_Facility table in an audit table (MissingFacilityIds)
	Conditional Split	Conditional split that separates Active and NonActive offenders
	Lookup	Lookup transform that captures the DormBed_ID column: <pre>SELECT rtrim(BKOBISFacility) as BKOBISFacility, rtrim(DormBed_ID) as DormBed_ID, rtrim(BKOBISBed) as BKOBISBed, ltrim(BedType) as BedType FROM DIM_DORMBED</pre>
	Union All 1	Brings together Active and NonActive Offenders (Active Offenders will have a DormBed_ID.)
	Gang Lookup	Lookup transform that retrieves Gang ID from the Dim_Gang table
	Is New Row?	This Lookup transform finds out if any rows that are loaded are newly entered: <pre>SELECT OffenderKey, Offender_ID, LastTransactionalRowDate FROM DIM_Offender WHERE (RowEndDate IS NULL)</pre>
	InsertOffenderRow	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Row Changed	A conditional split that checks to see if it is not a new row, then whether it is just an old row that has been modified
	Multicast	Takes the changed rows and sends them to two different tables (Temp #Updates table and Dim_Offender)
	UpdateInmateRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	#Updates	Places updated rows in #Updates temp table
	Missing Facility	SQL task that places facility IDs with no matching records in the Dim_Facility table in an audit table (MissingFacilityIds)
	Union All	Joins newly entered offender rows and old rows that have been modified

Table 20: Components of the Dim_Offender load process task

Symbol	Data flow component	Description
	Insert Dim_Offender	Brings together needed columns to be placed in Dim_Offender table

Dim_Offender Mark deleted

As shown in Figure 74, the Dim_Offender Mark deleted data flow task uses the newly created Dim_Offender table from the DCDW database on the REDACTEDserver to load the #Updates temp table.

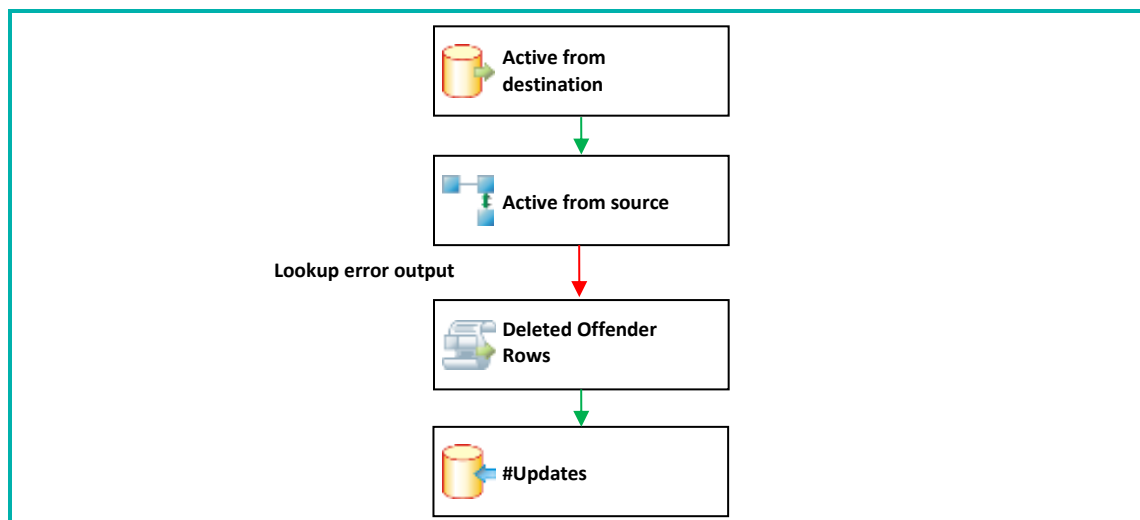


Figure 74: Dim_Offender Mark deleted data flow task

Table 21 describes the components of the Dim_Offender Mark deleted data flow task.

Table 21: Components of the Dim_Offender Mark deleted data flow task

Symbol	Data flow component	Description
	Active from Destination	Captures OffenderKey and Offender_ID columns from the table just created in the last data flow DIM_Offender from source database DCDW: <pre> SELECT OffenderKey, OffenderID FROM DIM_Offender WHERE (RowEndDate IS NULL) </pre>
	Active from Source	Lookup that captures OffenderKey to be placed in the temp table #Updates <pre> SELECT OffenderKey FROM dbo.v_dim_offender </pre>
	Deleted Offender Rows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes

Table 21: Components of the Dim_Offender Mark deleted data flow task

Symbol	Data flow component	Description
	#Updates	Brings together needed columns to be placed in #Updates temp table

Update EndDates

By running the following execute SQL statement, Update EndDates refreshes the RowEndDate column and UpdateAuditID columns with the latest data gathered from both DIM_Offender and #Updates:

```
Update i set RowEndDate=?,UpdateAuditID=?
FROM Dim_offender i
JOIN #Updates u on u.Offender_ID=i.Offender_ID
```

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 22 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 22: Connections for the Audit PostExecute of Package task

Connection	Target server	Target database	Configuration file
 SourceDCIS	REDACTED	DCIS	\\REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 Audit_DB	REDACTED	Audit_DB	\\REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW	REDACTED	DCDW	\\REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 Destination Temp	REDACTED	DCDW	\\REDACTED\config\SQL\SSIS\DestinationUpdates.dtsConfig

StaffDetail

As shown in figures 75 and 76, the StaffDetail package updates the Dim_Staff table in the DCDW data warehouse located on the REDACTEDserver.

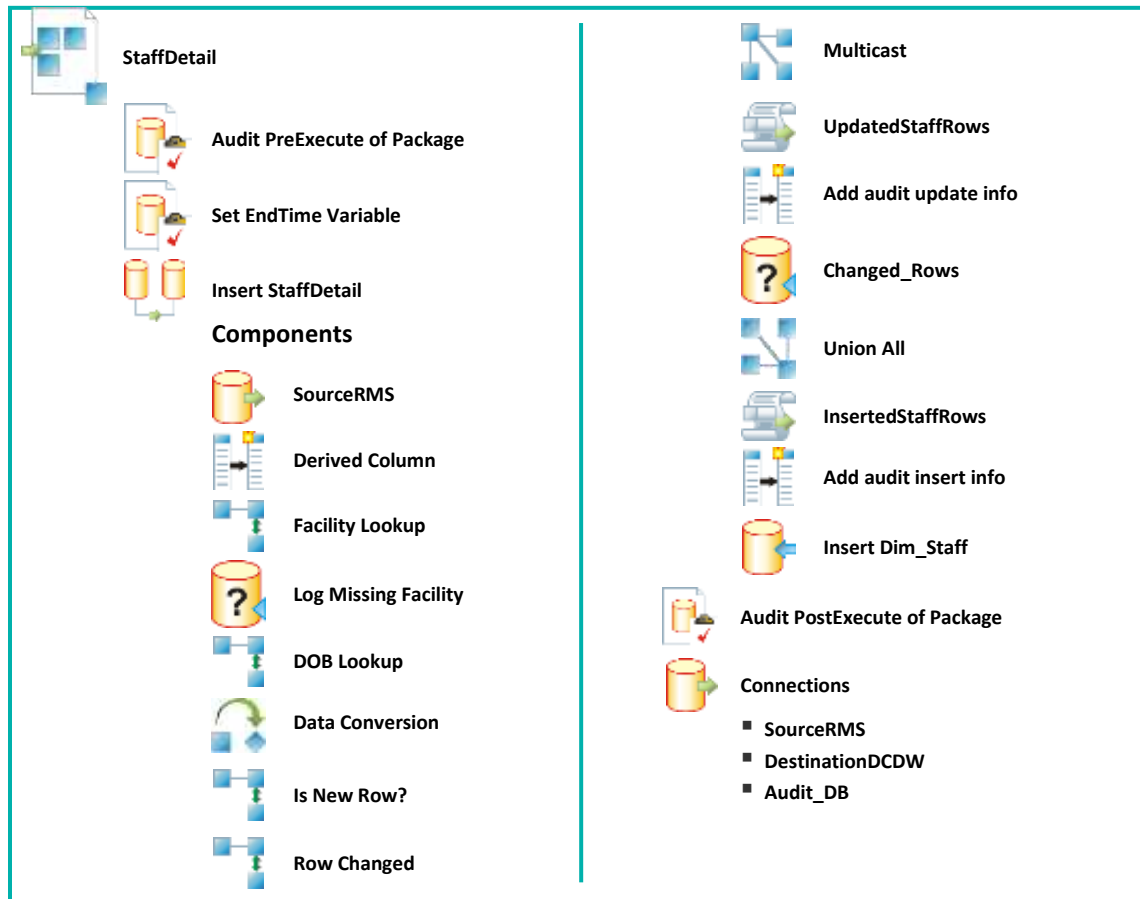


Figure 75: Contents of the StaffDetail package

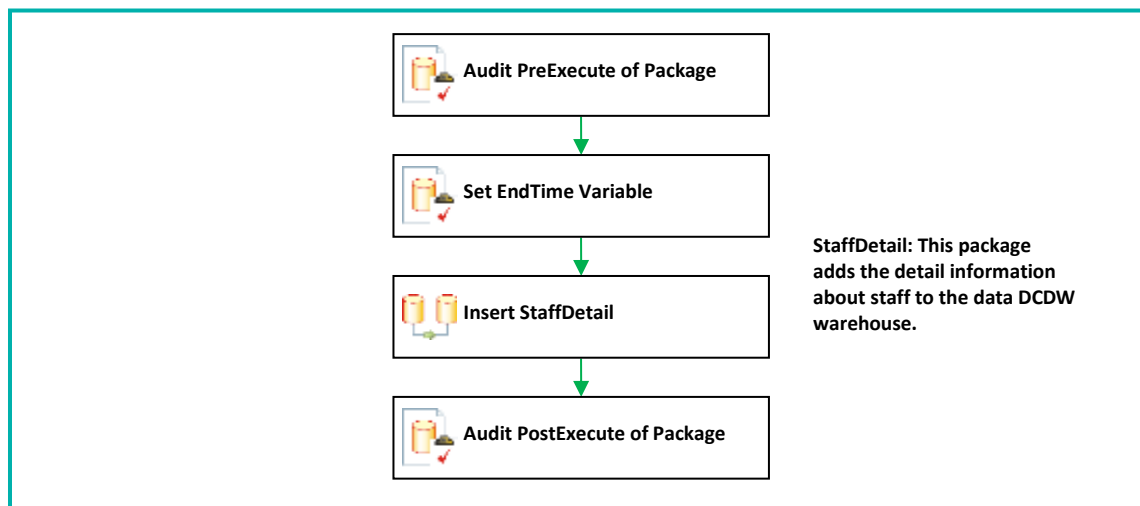


Figure 76: Control flow of the StaffDetail package

Audit PreExecute of Package

As shown in Figure 77, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

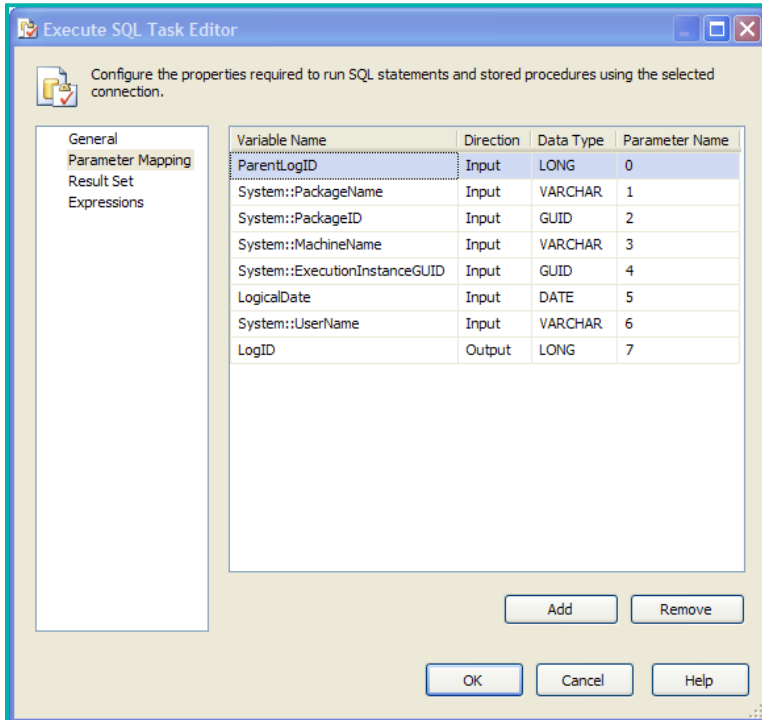


Figure 77: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Insert StaffDetail

As shown in Figure 78, the Insert StaffDetail data flow task uses the table MCIT0001 from the RMS database on the REDACTED server to update the DIM_Staff table in the DCDW data warehouse. This task also adds facility and date information related to each staff members to the DIM_Staff table, which originates from the DIM_Facility and DIM_DATE tables in the DCDW database on the REDACTED server.

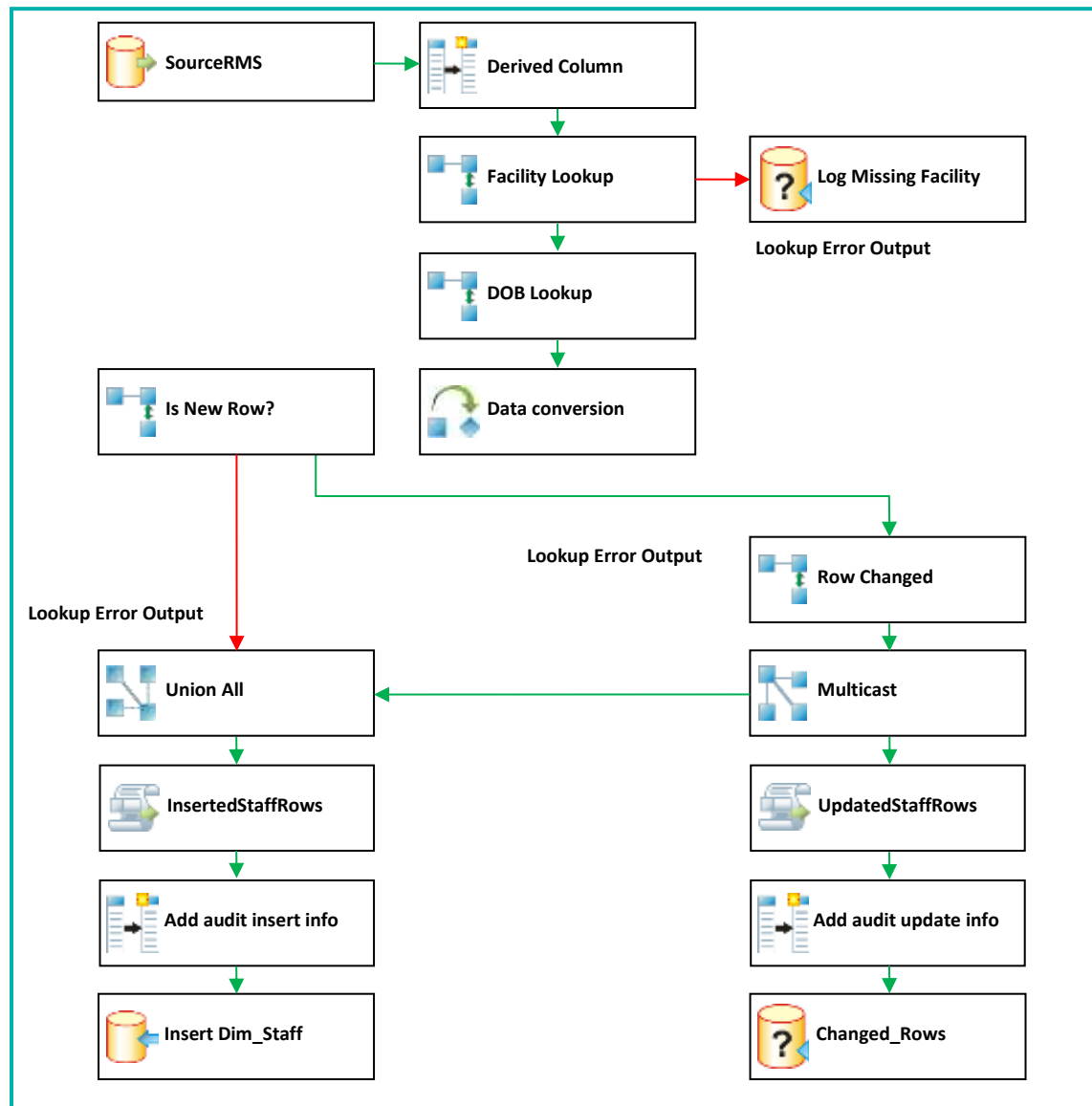


Figure 78: Insert StaffDetail data flow task

Table 23 describes the components of the Insert StaffDetail data flow task.

Table 23: Components of the Insert StaffDetail data flow task		
Symbol	Data flow component	Description
	SourceRMS	Captures columns from the stored procedure usp_DWStaff from source database RMS.
	Derived Column	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database
	Facility Lookup	Lookup transform that finds the facility each staff member is associated with the following SQL statement: <pre>SELECT Facility_ID, left(BKOBISFacility,4)As BKOBISFacility , Facility FROM DIM_Facility WHERE rowenddate is null</pre>
	Log Missing Facility	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIds)
	DOB Lookup	Lookup transform that captures the date of birth attribute for staff: <pre>SELECT Date_ID FROM DIM_DATE</pre>
	Data conversion	Converts the data type of the column CurrentlyTraining
	IS New Row?	This Lookup transform finds out if any rows that are loaded are newly entered
	Row Changed	Lookup checks to see if the row is not new has it been modified
	Multicast	Sends the results of the lookup to update command and Dim_Staff table
	UpdatedStaffRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Update Info	Derived column transform that adds UpdateAudit_ID and RowEndDate columns
	Changed_Rows	Places updated rows in #Updates temp table
	Union All	Joins newly entered staff rows and old rows that have been modified
	InsertedStaffRow	Custom Script Transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Insert Info	Derived column transform that adds InsertAudit_ID and RowStartDate columns
	Insert Dim_Staff	Brings together needed columns to place in DIM_Staff table

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 24 lists the target servers, target databases and configuration files for the Audit PostExecute of task.

Table 24: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
	SourceRMS	REDACTED	RMS	REDACTED\config\SQL\SSIS\SourceRMS.dtsConfig
	DestinationDCDW	REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
	Audit_DB	REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig

Dim_ExternalMovements

As shown in figures 79 and 80, Dim_ExternalMovements updates the Dim_ExternalMovements table in the DCDW data warehouse located on the REDACTED server.

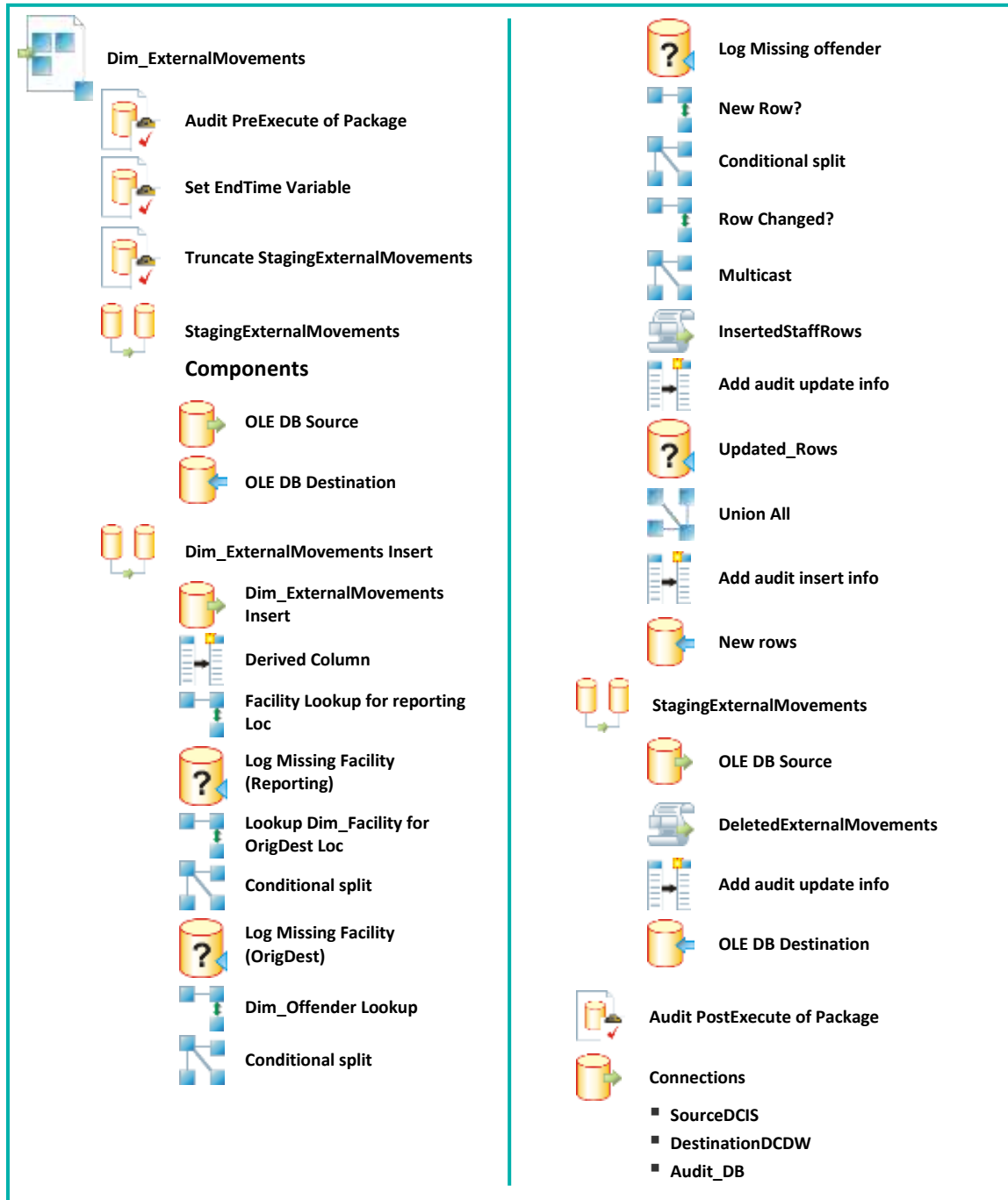


Figure 79: Contents of the ExternalMovements package

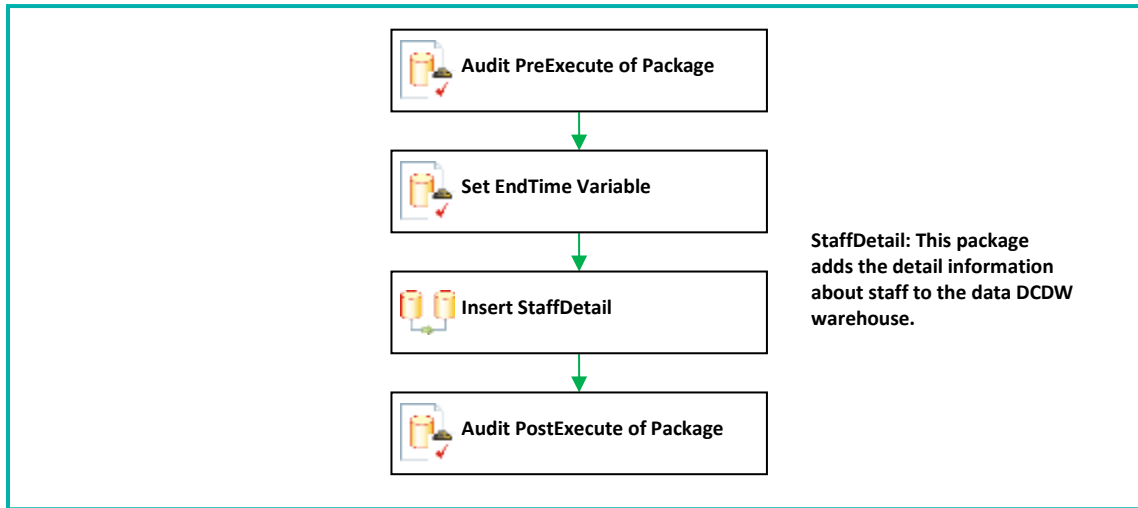


Figure 80: Control flow for the ExternalMovements package

Audit PreExecute of Package

As shown in Figure 81, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

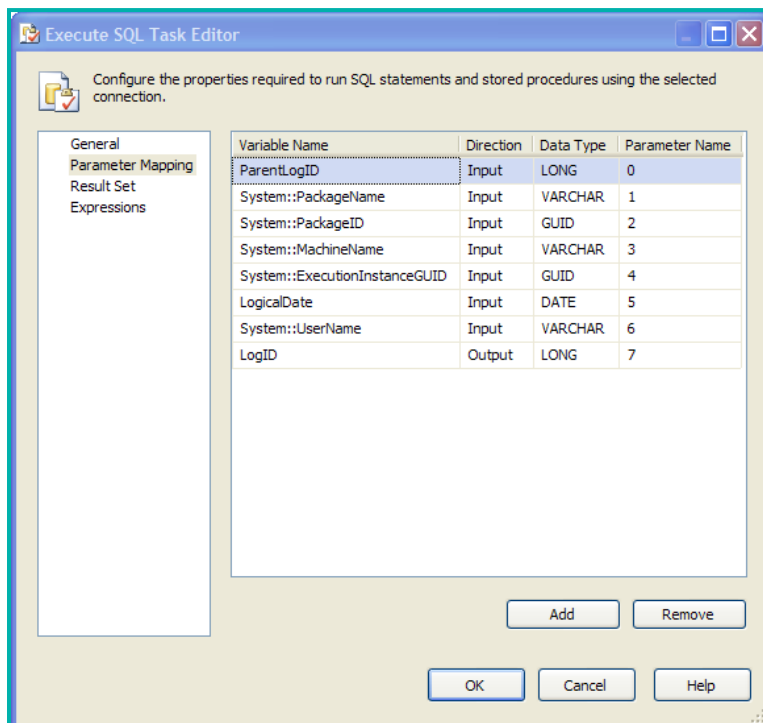


Figure 81: Execute SQL Task Editor dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Insert ExternalMovements

Truncate table StagingExternalMovements.

StagingExternalMovements

StagingExternalMovements uses EXEC usp_Dim_ExternalMovements to pull columns from DCIS to DCDW databases.

Dim_ExternalMovements Insert

As shown in Figure 82, the Dim_ExternalMovements Insert data flow task uses the table StagingExternalMovement from the DCDW database to update the Dim_ExternalMovements table. This task also adds missing facility and missing offender information to the Audit_DB.

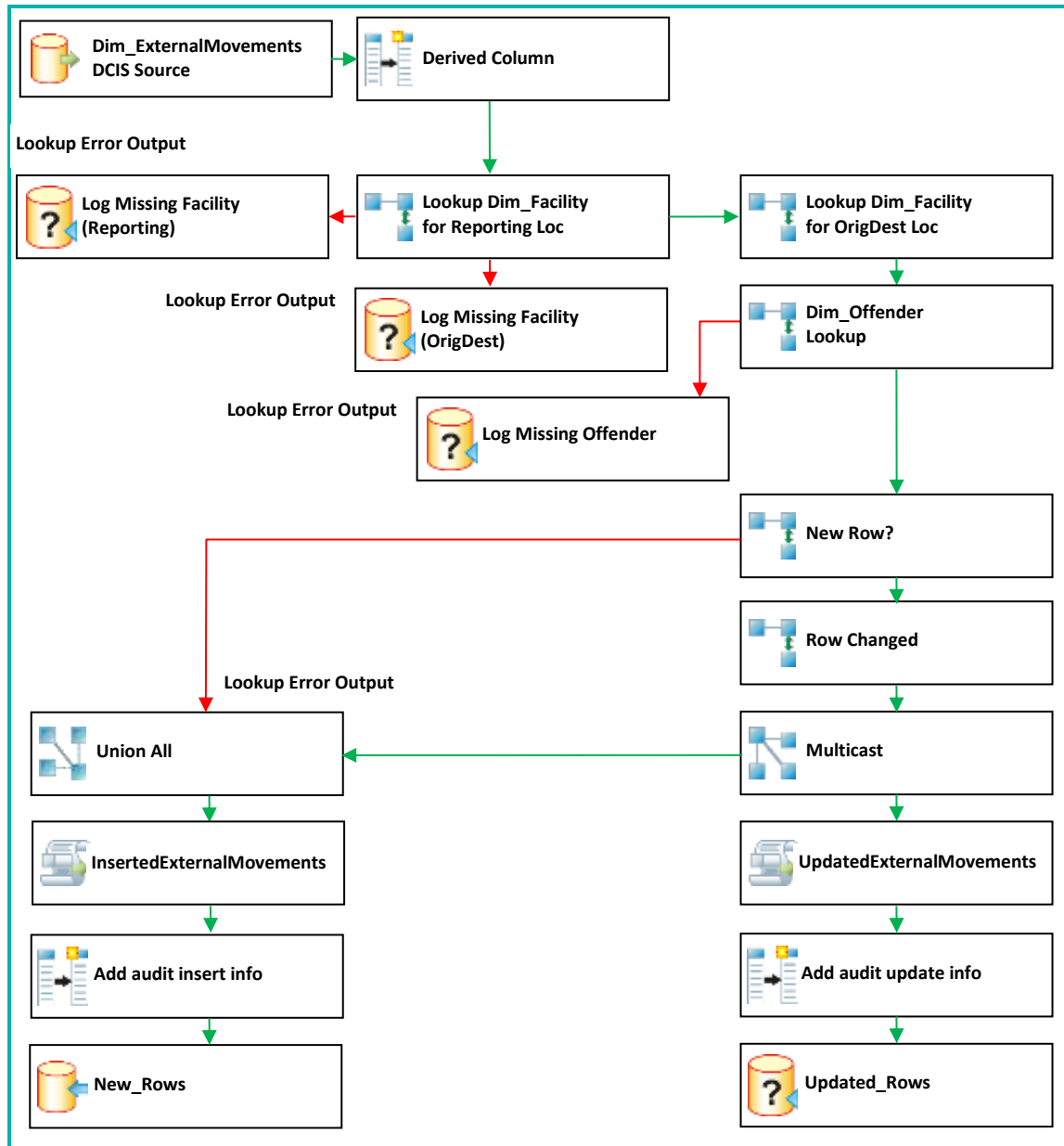


Figure 82: Dim_ExternalMovements Insert data flow task

Table 25 describes the components of the Dim_ExternalMovements Insert data flow task.

Table 25: Components of the ExternalMovements Insert data flow task		
Symbol	Data flow component	Description
	Dim_ExternalMovements DCIS Source	<code>SELECT * from StagingExternalMovements</code>
	Derived Column	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database
	Lookup Dim_Facility for Reporting Loc	Lookup transform that finds the facility each staff member is associated with the following SQL statement: <code>SELECT BKOBISFacility, Facility_ID FROM DIM_Facility WHERE rowenddate is null</code>
	Log Missing Facility (Reporting)	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIds)
	Lookup Dim_Facility for OrigDest Loc	Lookup transform that captures the date of birth attribute for staff: <code>SELECT BKOBISFacility, Facility_ID FROM DIM_Facility WHERE rowenddate is null</code>
	Conditional split	Successful match on Facility ID goes to Dim_Offender Lookup, while error output goes to MissingFacilityIDs table in Audit_DB database
	Dim_Offender Lookup	<code>SELECT BKOBISFacility, Facility_ID FROM DIM_Facility WHERE rowenddate is null</code>
	Conditional split	Successful match on BKDCNumber, MovementDate and MovementTime goes to Row Changed? lookup transform, while error output goes to Union All table transform as a new row
	Log Missing Offender	Places facilities that do not match Dim_Offender table in an audit table (MissingOffenders)
	New Row?	This Lookup transform finds out if any rows that are loaded are newly entered
	Conditional split	Successful match on Offender_ID goes to New Row? lookup transform Lookup, while Error output goes to MissingOffenders table in Audit_DB database
	Row Changed?	Lookup checks to see if the row is not new has it been modified
	Multicast	Sends the results of the lookup to update command and Dim_ExternalMovements table

Table 25: Components of the ExternalMovements Insert data flow task

Symbol	Data flow component	Description
	UpdatedExternalMovements	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Update Info	Derived column transform that adds UpdateAudit_ID and RowEndDate columns
	Updated_Rows	Update Dim_ExternalMovements Set UpdateAudit_ID = ?, RowEndDate = ? Where Offender_ID = ? and MovementDate = ? and MovementTime = ?
	Union All	Joins newly entered staff rows and old rows that have been modified
	InsertedExternalMovements	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Insert Info	Derived column transform that adds InsertAudit_ID and RowStartDate columns
	New_Rows	Brings together needed columns to place in Dim_ExternalMovements table

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 26 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 26: Connections for the Audit PostExecute of Package task

Connection	Target server	Target database	Configuration file
 SourceDCIS	REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 DestinationDCDW	REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 Audit_DB	REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig

Dim_HighProfile_Offender

As shown in figures 83 and 84, the Dim_HighProfile_Offender SSIS package updates the Dim_HighProfile_Offender table in the DCDW data warehouse on the REDACTEDserver. In addition, it adds to the ExecutionLog table of the Audit_DB database on the same server.

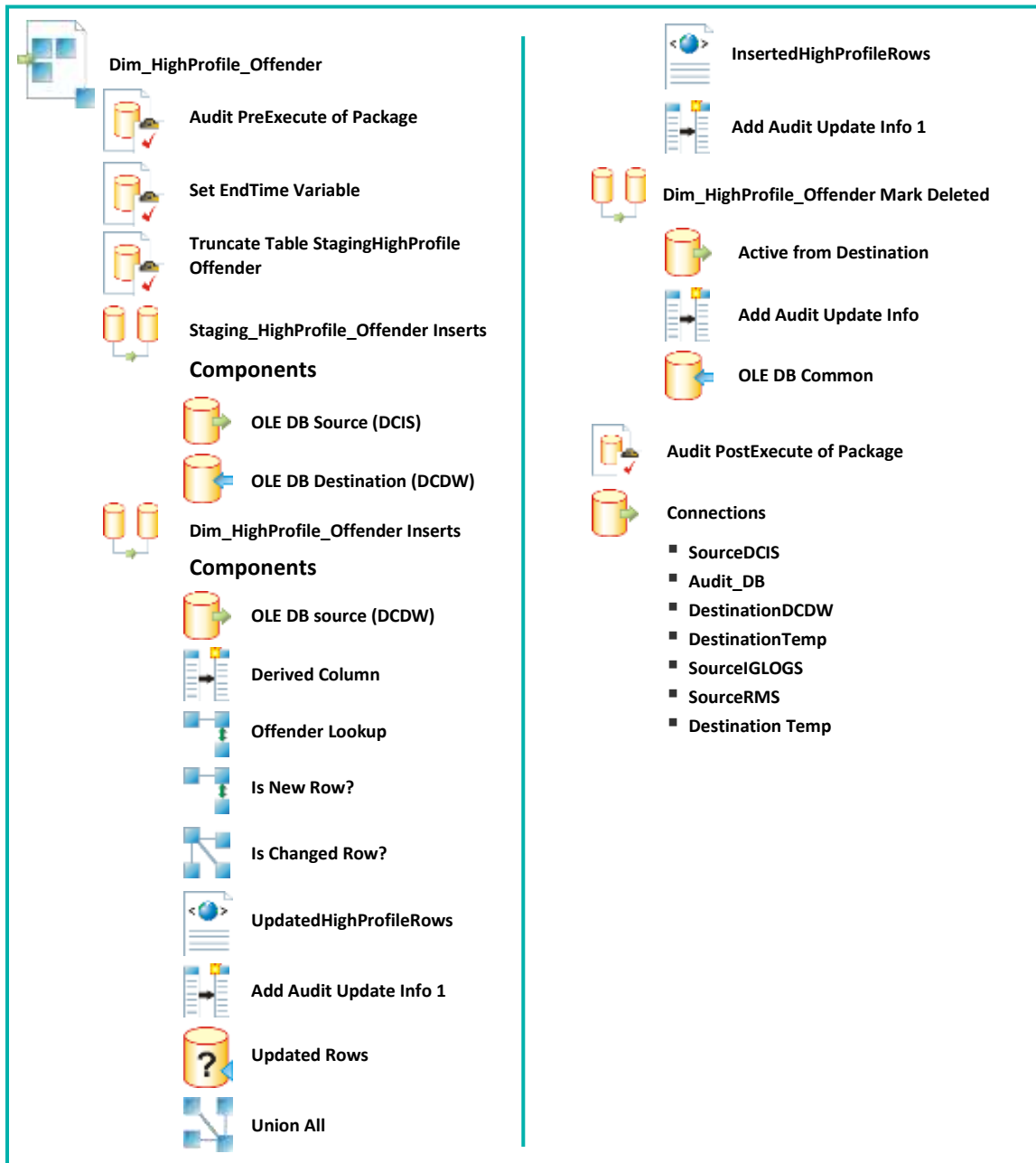


Figure 83: Contents of the Dim_HighProfile_Offender package

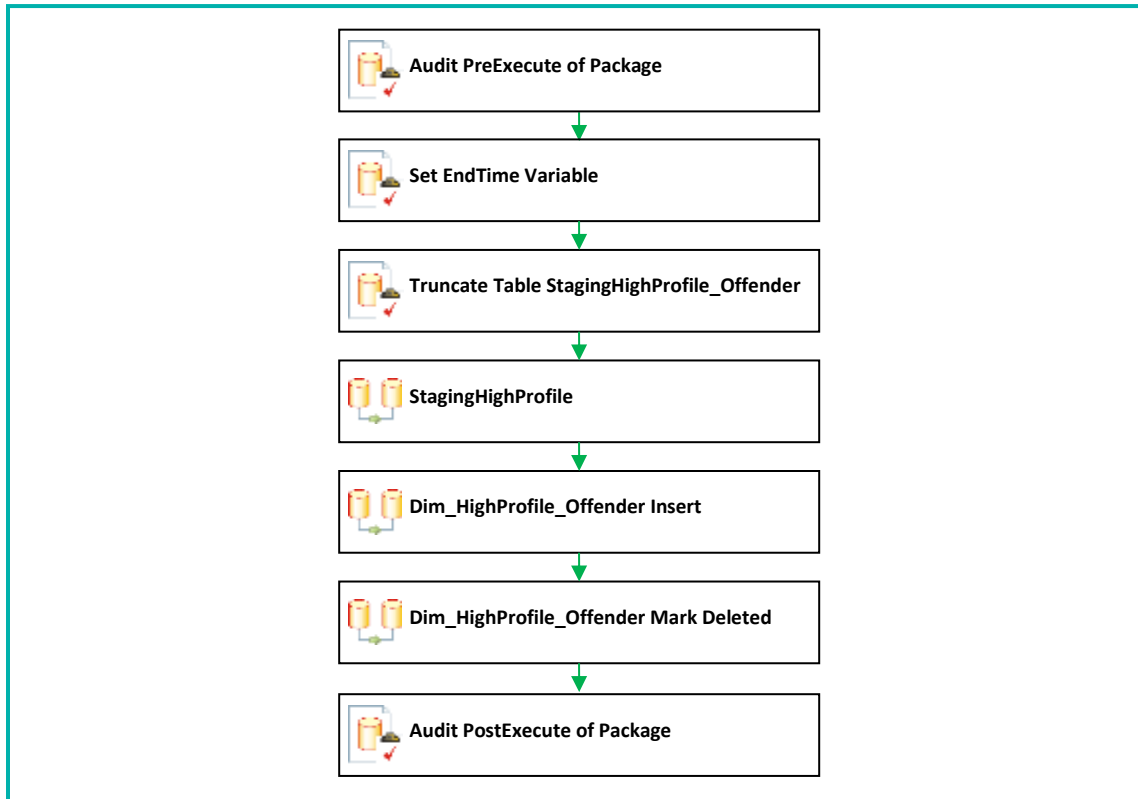


Figure 84: Control flow for the Dim_HighProfile_Offender package

Audit PreExecute of Package

As shown in Figure 85, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

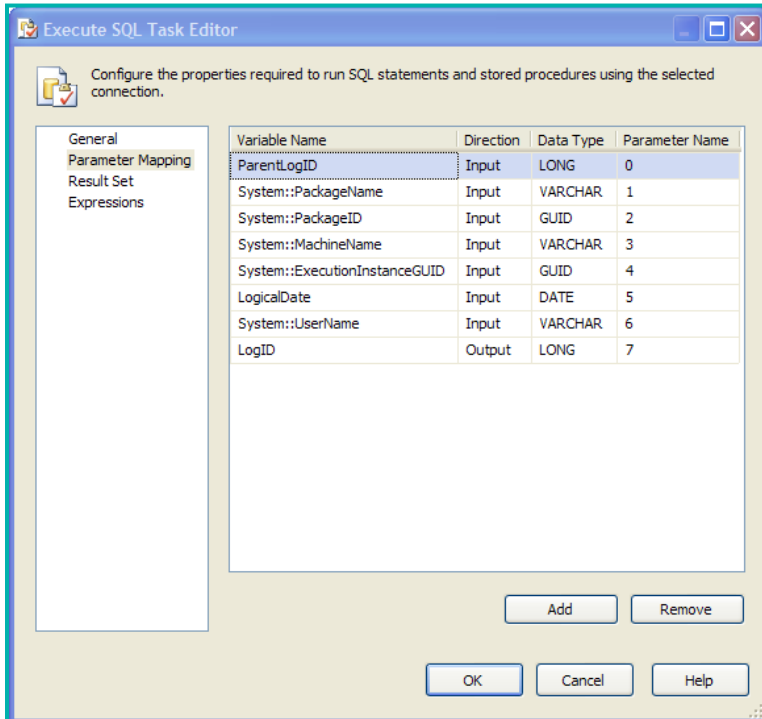


Figure 85: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Create Temp Table #Updates

By running the following execute SQL statement, Create Temp Table #Updates creates a temporary table on the REDACTED server to hold offender data rows that have been changed:

```
CREATE TABLE [#Updates]
(
    [Offender_ID] INTEGER
)
```

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_HighProfile_Offender Inserts

As shown in Figure 86, the Dim_HighProfile_Offender Inserts data flow task selects records from the StagingHighProfile_Offender table in the DCDW database on the REDACTED server and inserts them into the Dim_HighProfile_Offender table in the data warehouse.

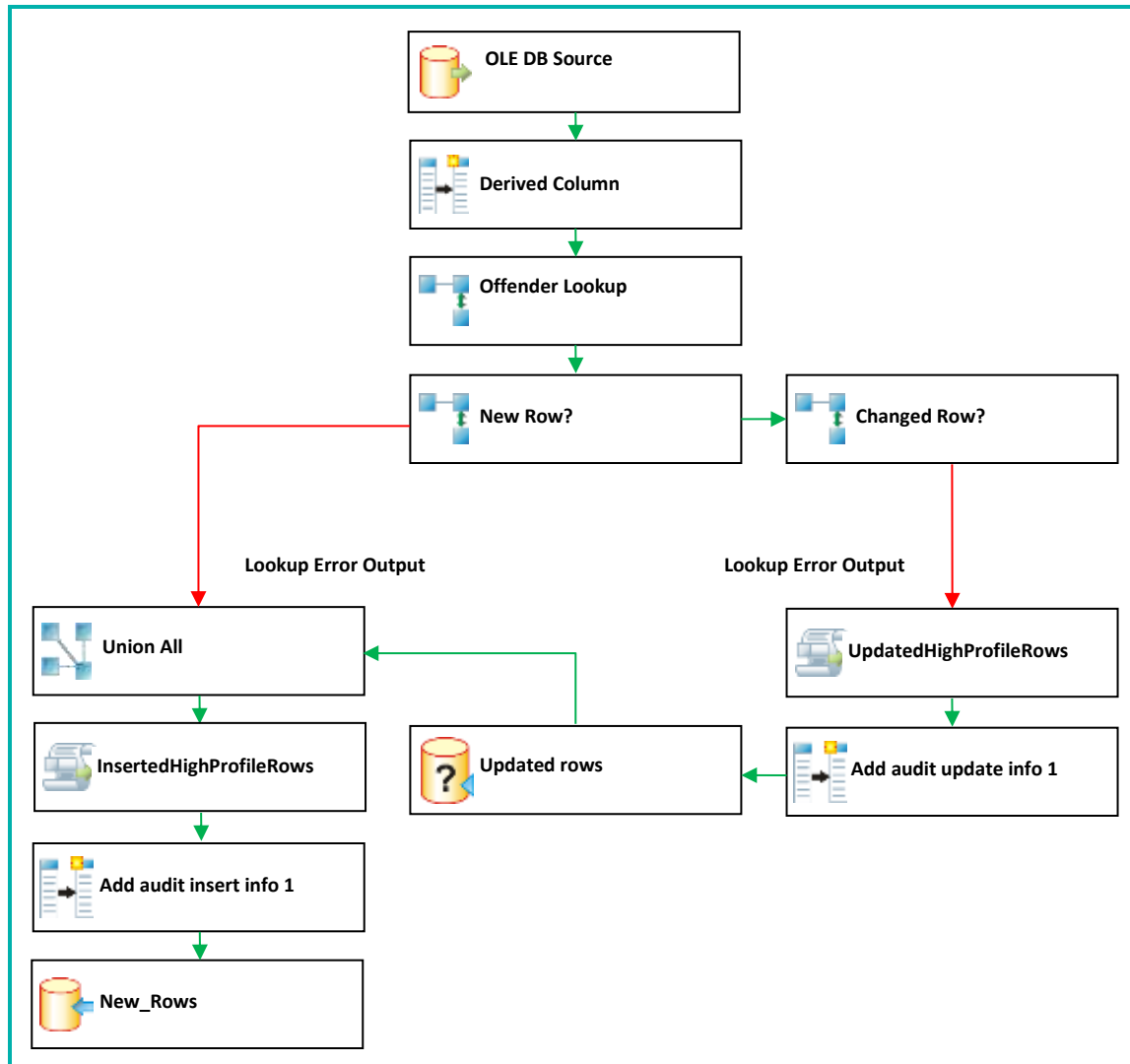


Figure 86: Dim_HighProfile_Offender Inserts data flow task

Table 27 describes the components of the Dim_HighProfile_Offender Inserts data flow task.

Table 27: Components of the Dim_HighProfile_Offender Inserts data flow task		
Symbol	Data flow component	Description
	OLE DB Source	Captures columns from the stored procedure StagingHighProfile_Offender from source database DCDW
	Derived Column	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database
	Offender Lookup	Lookup transform that captures the Offender_ID column: <pre> SELECT max(Dim_Offender.Offender_ID) as Offender_ID, Dim_Offender.BKDCNumber FROM StagingHighProfile_Offender INNER JOIN Dim_Offender ON StagingHighProfile_Offender.bkdcnumber = Dim_Offender.bkdcnumber WHERE dim_offender.rowenddate is null GROUP BY Dim_Offender.BKDCNumber </pre>
	Next Row?	This lookup transform finds out if any rows that are loaded are newly entered: <pre> SELECT bkdcnumber,CriticalCaseDate,CriticalCaseTi me FROM dim_highprofile_offender WHERE rowenddate is null </pre>
	Changed Row	This lookup uses the error output for records that have had their column values change
	UpdatedHighProfileRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Update Info 1	Derived column transform that adds UpdateAudit_ID and RowEndDate columns in to changed records
	SQL Command	Executes an update command that updates UpdateAudit_ID and RowEndDate for changed records: <pre> Update Dim_HighProfile_Offender Set UpdateAudit_ID = ?, RowEndDate = ? Where BKDCNumber = ? and CriticalCaseDate = ? and CriticalCaseTime =? </pre>
	Union All	Joins newly entered offender rows and old rows that have been modified
	InsertedHighProfileRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Insert Info	Derived column transform that adds InsertAudit_ID and RowStartDate columns for new records
	New_Rows	Inserts Union All records into Dim_HighProfile_Offender table

Dim_HighProfile_Offender Mark deleted

As shown in Figure 87, the Dim_HighProfile_Offender Mark deleted task uses the Dim_HighProfile_Offender table from the DCIS database.

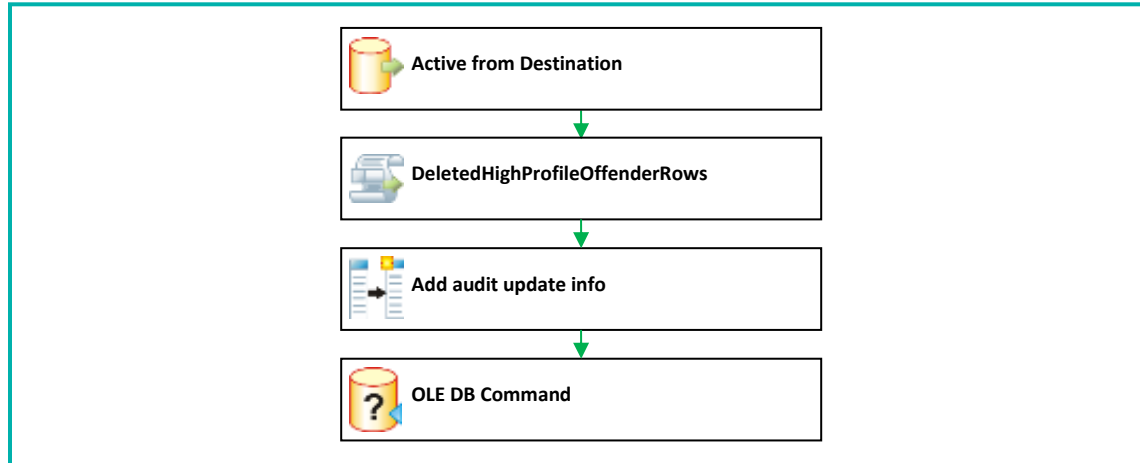


Figure 87: Control flow for the Dim_HighProfile_Offender package

Table 28 describes the components of the Dim_HighProfile_Offender control flow package.

Table 28: Components of the Dim_HighProfile_Offender control flow package		
Symbol	Data flow component	Description
	Active from Destination	Captures active HighProfileOffender rows from the DCDW database that no longer exist in the source database
	Deleted HighProfileOffender Rows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Update Info	Derived column transform that adds UpdateAudit_ID and RowEndDate columns
	OLE DB Command	This execute SQL task updates the RowEndDate column and UpdateAuditID columns from data gathered from Dim_HighProfile_Offender and #Updates and runs the following statement: <pre>Update Dim_HighProfile_Offender Set UpdateAudit_ID = ?, RowEndDate = ? Where BKDCNumber = ? and CriticalCaseDate = ? and CriticalCaseTime =?</pre>

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 29 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 29: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceDCIS		REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 SourceIGLOGS		REDACTED	STG	REDACTED\config\SQL\SSIS\SourceIGLOGS.dtsConfig
 SourceRMS		REDACTED	RMS	REDACTED\config\SQL\SSIS\SourceRMS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 Destination Temp		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationUpdates.dtsConfig

Dim_Health_Offender

As shown in figures 88 and 89, the Dim_Health_Offender package updates the Dim_Health_Offender table in the DCDW data warehouse on the REDACTED server. In addition, it adds to the ExecutionLog table of the Audit_DB database on the same server.

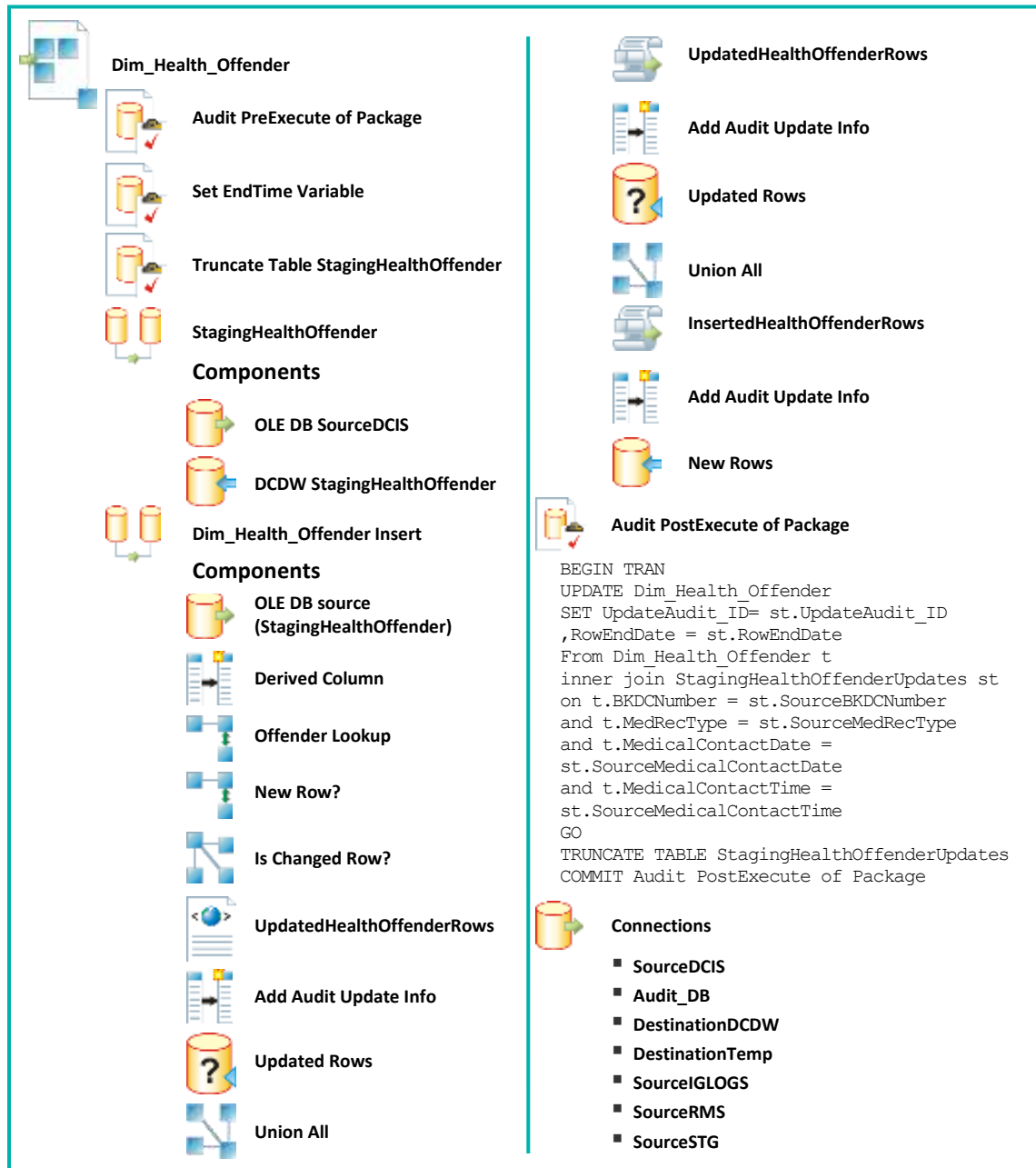


Figure 88: Contents of the Dim_Health_Offender package

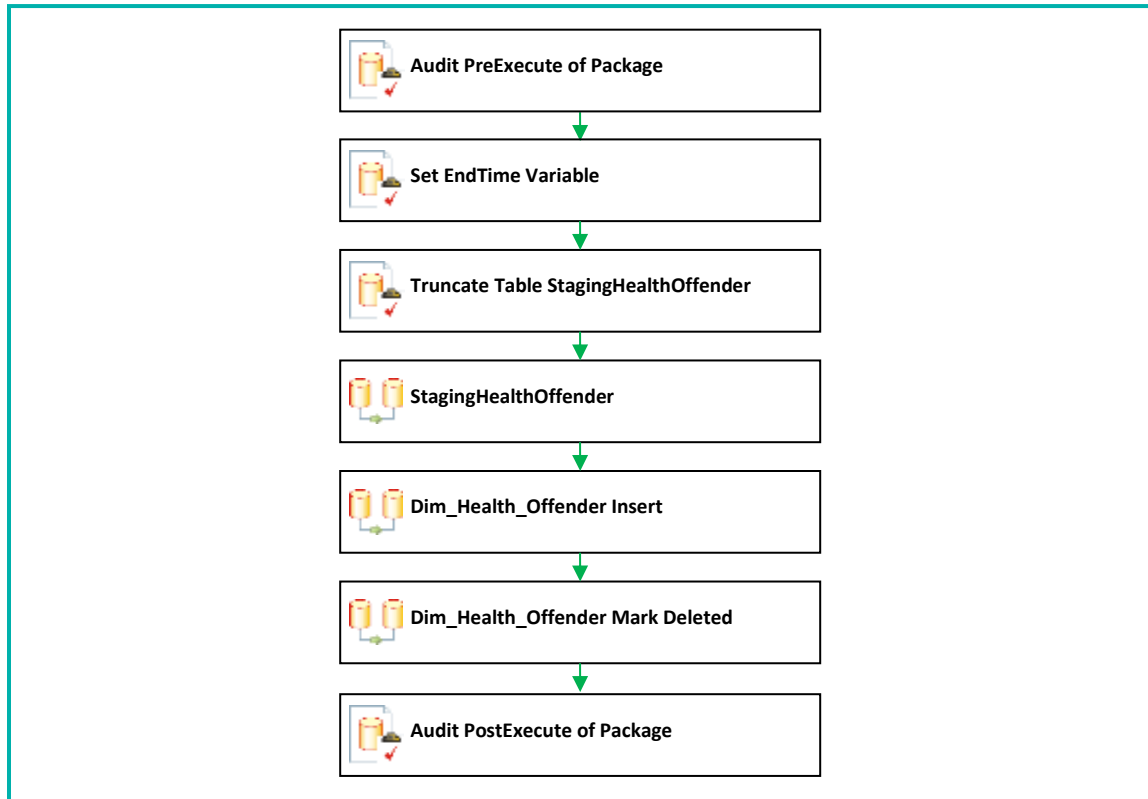


Figure 89: Control flow for the Dim_Health_Offender package

Audit PreExecute of Package

As shown in Figure 85, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

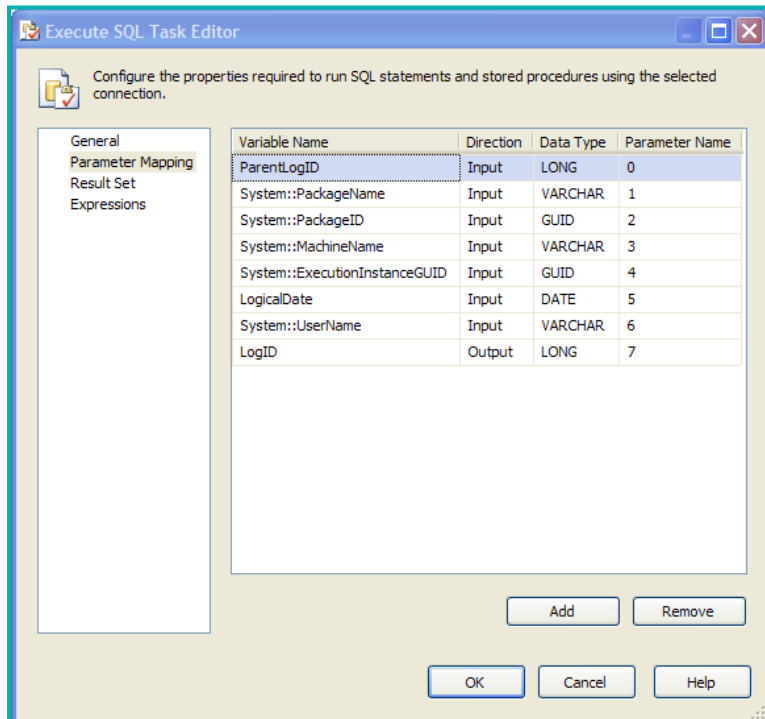


Figure 90: Execute SQL Task Editor dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Truncate Table StagingHealthOffender

```
Truncate Table StagingHealthOffender.
```

StagingHealthOffender

```
exec usp_Health_Offender ?
```

Dim_Health_Offender Insert

As shown in Figure 91, the Dim_Health_Offender Insert data flow task selects records from the StagingHighProfile_Offender table in the DCDW database on the REDACTED server and inserts them into the Dim_Health_Offender table in the data warehouse.

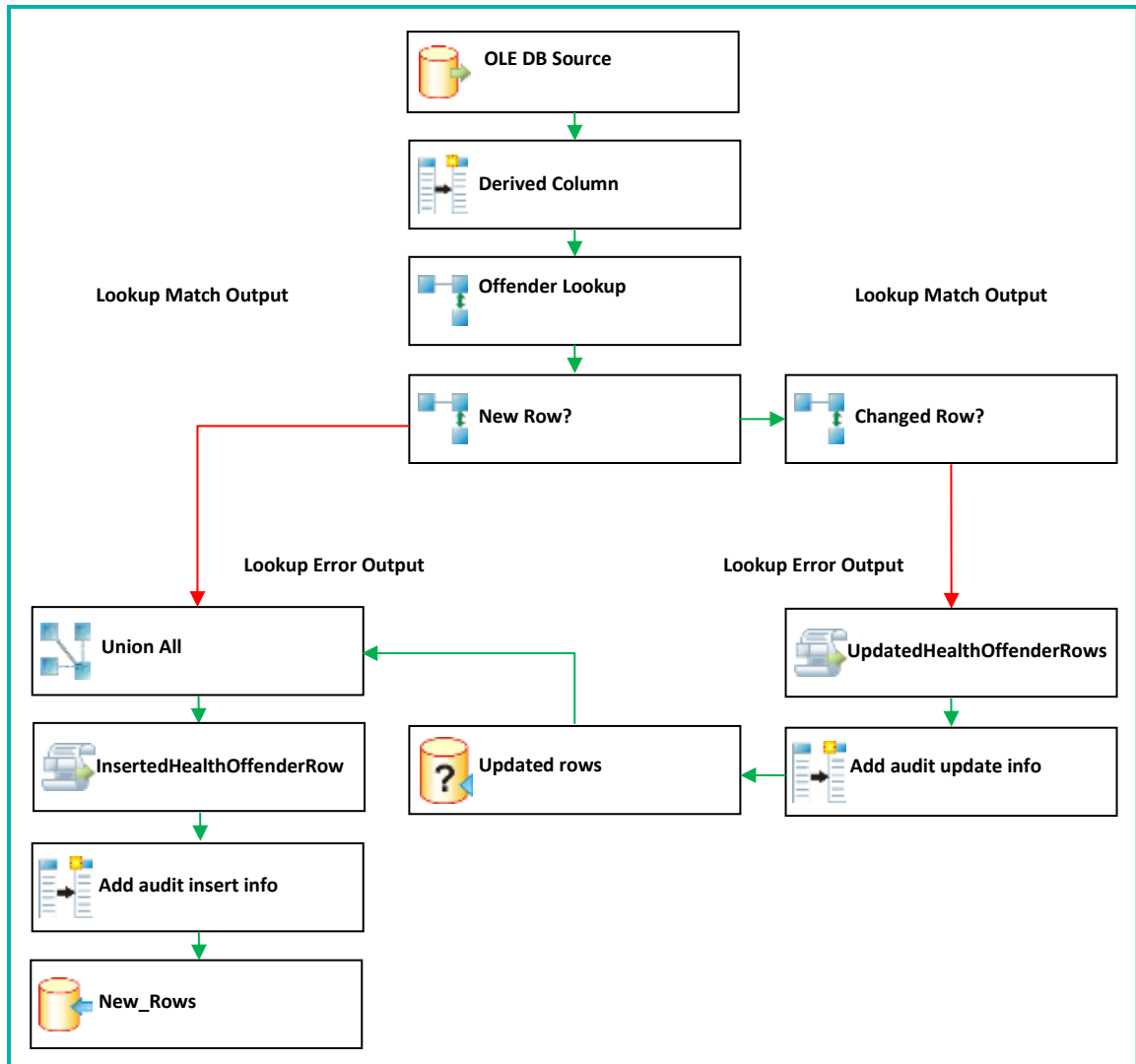


Figure 91: Dim_Health_Offender Inserts data flow task

Table 30 describes the components of the Dim_Health_Offender Inserts data flow task.

Table 30: Components of the Dim_Health_Offender Inserts data flow task		
Symbol	Data flow component	Description
	OLE DB Source	Captures columns from the stored procedure StagingHealthOffender from source database DCDW
	Derived Column	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database

Table 30: Components of the Dim_Health_Offender Inserts data flow task

Symbol	Data flow component	Description
	Offender Lookup	Lookup transform that captures the Offender_ID column: <pre> SELECT max(Dim_Offender.Offender_ID) as Offender_ID, Dim_Offender.BKDCNumber FROM StagingHealthOffender INNER JOIN Dim_Offender ON StagingHealthOffender.bkdcnumber = Dim_Offender.bkdcnumber WHERE dim_offender.rowenddate is null GROUP BY Dim_Offender.BKDCNumber </pre>
	Next Row?	This lookup transform finds out if any rows that are loaded are newly entered: <pre> SELECT BKDCNumber,MedicalContactDate, MedicalContactTime, MedRecType FROM dim_health_offender WHERE rowenddate is null </pre>
	Changed Row	This Lookup transform finds out whether rows loaded into the database have changed by using the error output for rows where all columns don't match: <pre> SELECT Offender_ID ,BKDCNumber ,PulhesScoreP ,PulhesScoreU ,PulhesScoreL ,PulhesScoreH ,PulhesScoreE ,PulhesScoreS ,PulhesScoreD ,PulhesScoreX ,PulhesScoreI ,PulhesScoreT ,OverallMedicalGrade ,LastMedicalGradeDate ,LastMedicalGradeTime ,OverallMedicalLevel ,SingleDoseIndicator ,PhysicalImpairment ,HearingImpairment ,VisionImpairment ,DevelopmentImpairment FROM Dim_Health_Offender where rowenddate is null </pre>
	UpdatedHighProfileRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Update Info 1	Derived column transform that adds UpdateAudit_ID and RowEndDate columns in to changed records
	Update Rows	Inserts into StagingHealthOffenderUpdates

Table 30: Components of the Dim_Health_Offender Inserts data flow task		
Symbol	Data flow component	Description
	Union All	Joins newly entered offender rows and old rows that have been modified
	InsertedHighProfileRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Insert Info	Derived column transform that adds InsertAudit_ID and RowStartDate columns for new records
	New_Rows	Places updated rows in #Updates temp table

Dim_HighProfile_Offender Mark deleted

As shown in Figure 92, the Dim_Health_Offender Mark deleted data flow task uses the newly created Dim_Offender table from the DCDW database on the REDACTED server to load the #Updates temp table.

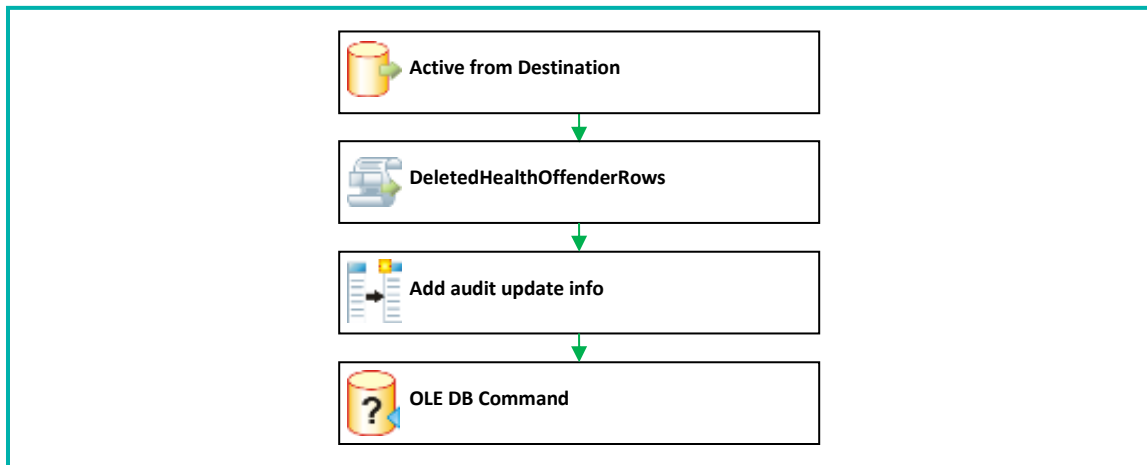


Figure 92: Control flow for the Dim_Health_Offender Mark deleted package

Table 31 describes the components of the Dim_Health_Offender Mark deleted control flow package.

Table 31: Components of the Dim_Health_Offender Mark deleted control flow package		
Symbol	Data flow component	Description
	Active from Destination	<pre> SELECT o.BKDCNumber FROM OBISHealthStatus hs Left Join OBISOffender o on o.offenderkey=hs.offenderkey WHERE hs.rowenddate >=? AND o.rowenddate is null AND not exists (select 1 FROM OBISHealthStatus AS hs2 Left Join OBISOffender o2 on o2.offenderkey=hs2.offenderkey WHERE o.BKDCNumber = o2.BKDCNumber) </pre>
	DeletedHealthOffender Rows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Update Info	Derived column transform that adds UpdateAudit_ID and RowEndDate columns
	OLE DB Command	<pre> Update rows marked deleted Update Dim_Health_Offender Set UpdateAudit_ID = ?, RowEndDate = ? Where BKDCNumber=? </pre>

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 32 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 32: Connections for the Audit PostExecute of Package task				
Connection	Target server	Target database	Configuration file	
 SourceDCIS	REDACTED	DCIS	\REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig	
 SourceSTG		STG	None	
 SourceIGLOGS	REDACTED	IGLOGS	REDACTED\config\SQL\SSIS\SourceIGLOGS.dtsConfig	

Table 32: Connections for the Audit PostExecute of Package task

Connection	Target server	Target database	Configuration file
 SourceRMS	REDACTED	RMS	REDACTED\config\SQL\SSIS\SourceRMS.dtsConfig
 Audit_DB	REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW	REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 Destination Temp	REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationUpdates.dtsConfig

Dim_ManagementScores_Offender

As shown in figures 93 and 94, the Dim_ManagementScores_Offender package updates the Dim_ManagementScores_Offender table in the in the DCDW data warehouse on the REDACTED server. In addition, it adds to the ExecutionLog table of the Audit_DB database, also located on the REDACTED server.

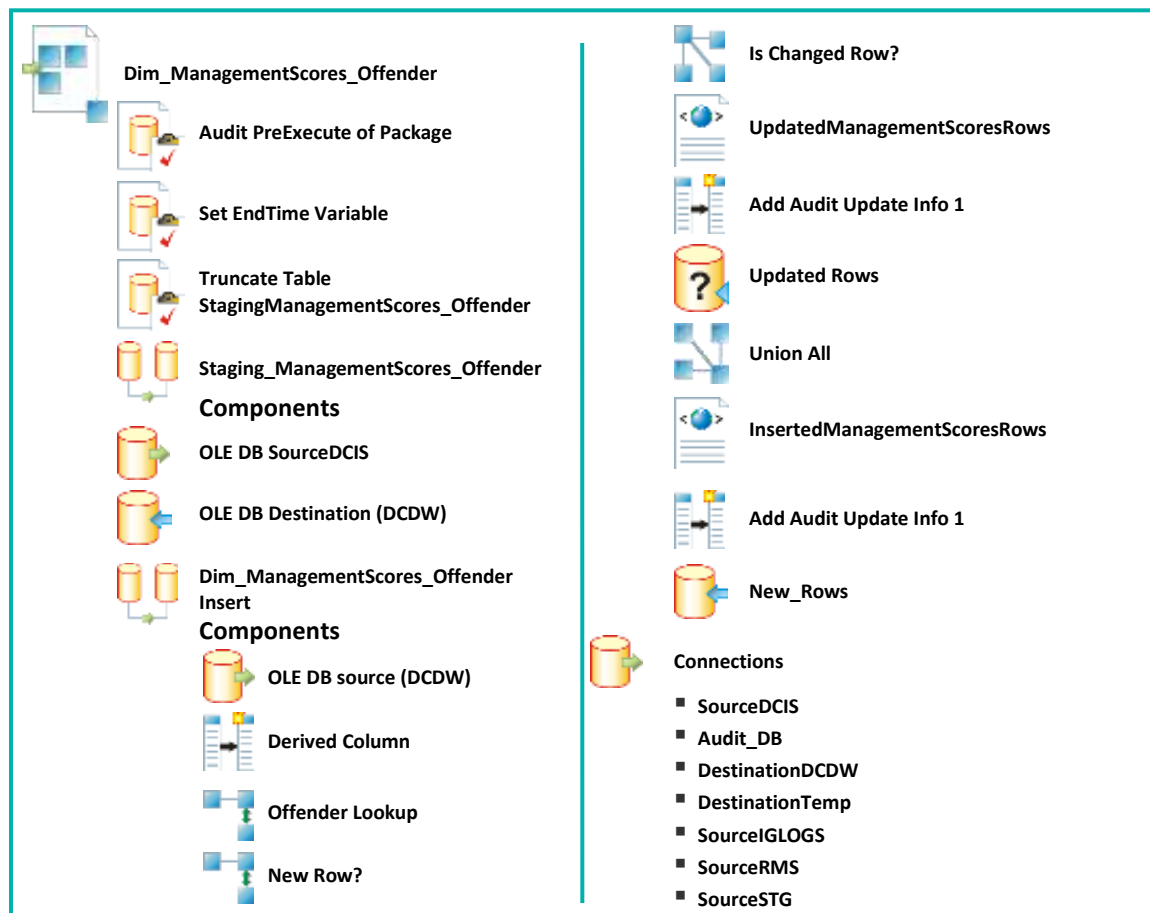


Figure 93: Contents of the Dim_ManagementScores_Offender package

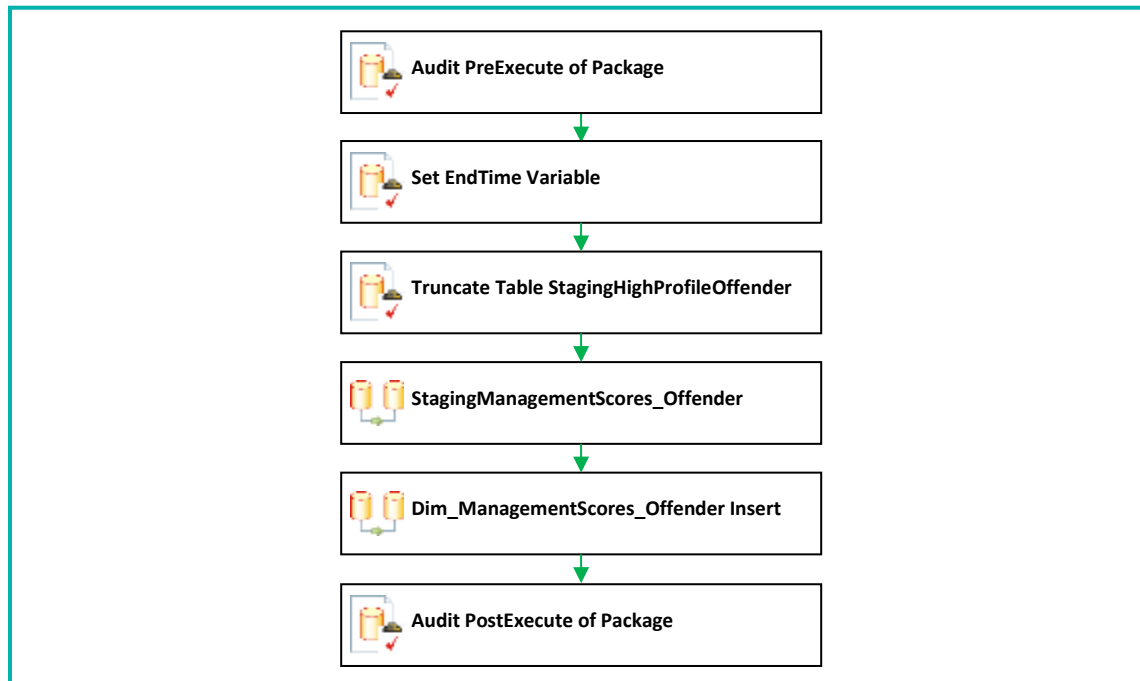


Figure 94: Control flow for the Dim_ManagementScores_Offender package

Audit PreExecute of Package

As shown in Figure 95, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

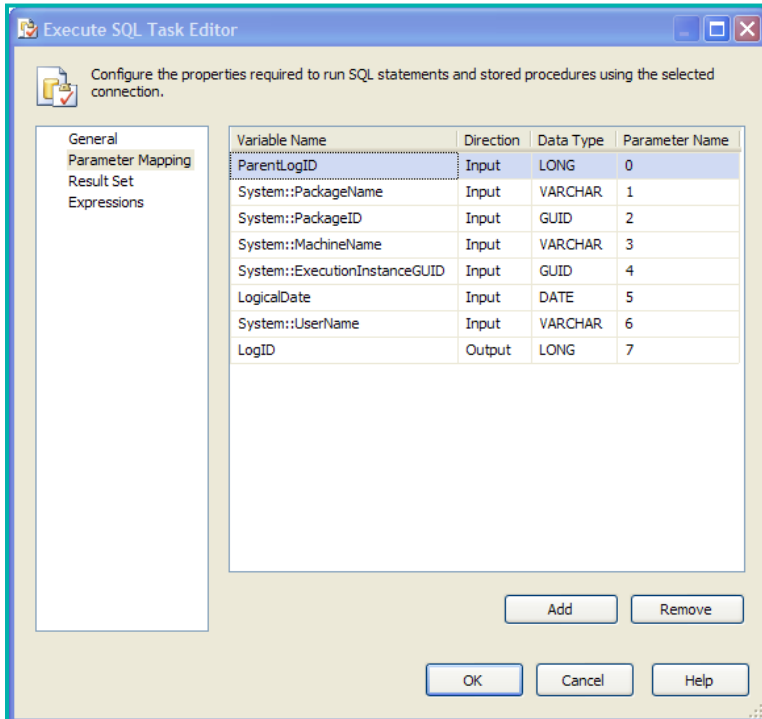


Figure 95: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Truncate Table StagingManagementScoreOffender

```
Truncate Table StagingManagementScoresOffender
```

StagingManagementScores_Offender

The StagingManagementScores_Offender data flow task executes usp_InternalManagement_Offender procedure in the DCIS database on the REDACTED and inserts them into the taggingManagementScoresOffender table in the DCDW database in the data warehouse.

Dim_ManagementScores_Offender Inserts

As shown in Figure 96, the Dim_ManagementScores_Offender Insert data flow task selects records from the StagingHighProfile_Offender table in the DCDW database on the REDACTED server and inserts them into the Dim_ManagementScores_Offender table in the data warehouse.

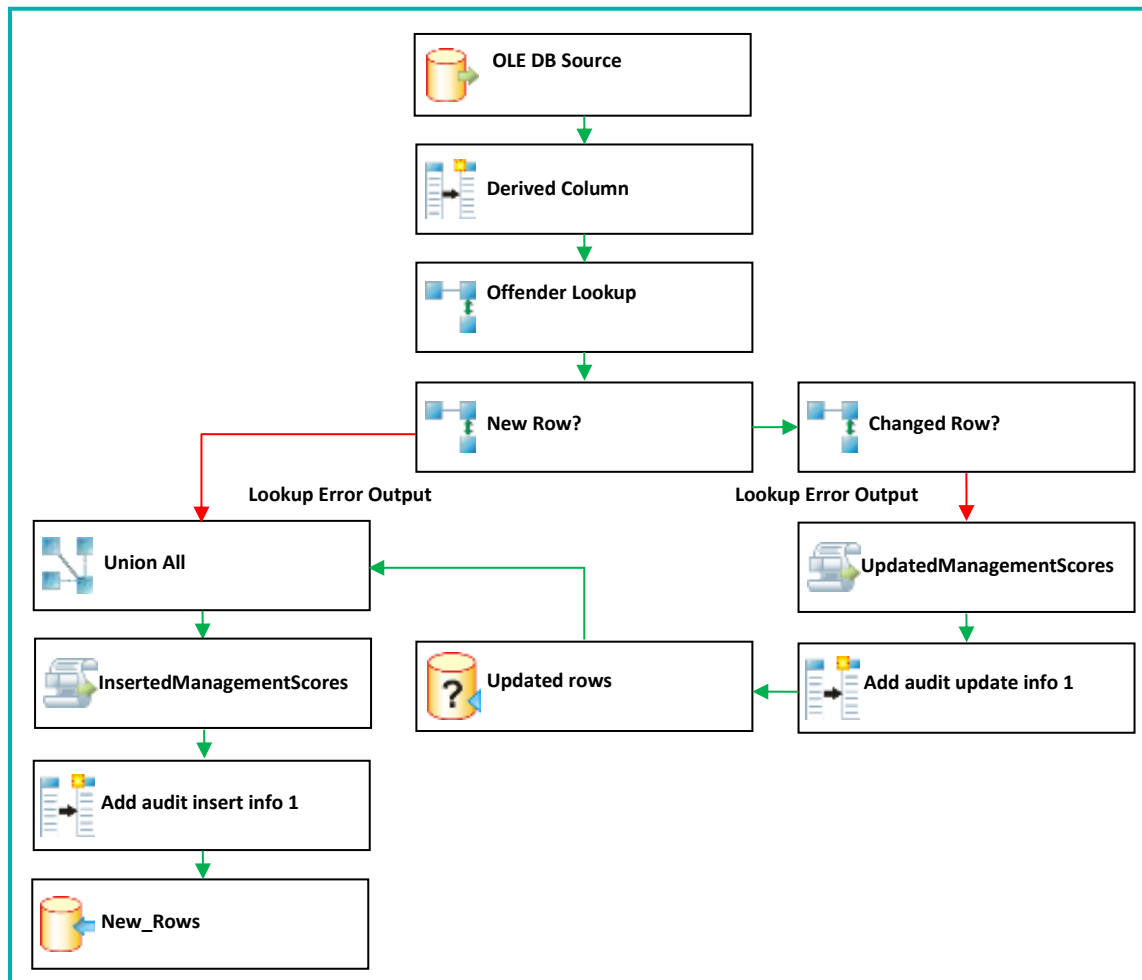


Figure 96: Dim_Health_Offender Inserts data flow task

Table 33 describes the components of the Dim_ManagementScores_Offender Inserts data flow task.

Table 33: Components of the Dim_ManagementScores_Offender Inserts data flow task		
Symbol	Data flow component	Description
	OLE DB Source	Captures columns from the stored procedure StagingManagementScores_Offender from source database DCDW
	Derived Column	Derived column transform that adds AuditLogId column in required tables in the Audit_DB database

Table 33: Components of the Dim_ManagementScores_Offender Inserts data flow task

Symbol	Data flow component	Description
	Offender Lookup	Lookup transform that captures the Offender_ID column: <pre> SELECT max(Dim_Offender.Offender_ID) as Offender_ID, Dim_Offender.BKDCNumber FROM StagingManagementScores_Offender INNER JOIN Dim_Offender ON ManagementScores_Offender.bkdcnumber = Dim_Offender.bkdcnumber WHERE dim_offender.rowenddate is null GROUP BY Dim_Offender.BKDCNumber </pre>
	Next Row?	This lookup transform finds out if any rows that are loaded are newly entered: <pre> SELECT BKDCNumber,MedicalContactDate, MedicalContactTime, MedRecType FROM dim_ManagementScores_offender WHERE rowenddate is null </pre>
	Changed Row	This Lookup uses the error output for records that have had their column values changed
	UpdatedHighProfileRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Update Info 1	Derived column transform that adds UpdateAudit_ID and RowEndDate columns in to changed records
	Update Rows	Executes the following update command that updates UpdateAudit_ID and RowEndDate for changed records: <pre> UPDATE Dim_ManagementScores_Offender SET UpdateAudit_ID = ?, RowEndDate = ? WHERE BKDCNumber = ? AND CriticalCaseDate=? AND CriticalCaseTime=? </pre>
	Union All	Joins newly entered offender rows and old rows that have been modified
	InsertedManagementScoresRows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	Add Audit Insert Info	Derived column transform that adds InsertAudit_ID and RowStartDate columns for new records
	New_Rows	Inserts Union All records into Dim_ManagementScores_Offender table

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 34 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 34: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceDCIS		REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 SourceIGLOGS		REDACTED	IGLOGS	REDACTED\config\SQL\SSIS\SourceIGLOGS.dtsConfig
 SourceRMS		REDACTED	RMS	REDACTED\config\SQL\SSIS\SourceRMS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 Destination Temp		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationUpdates.dtsConfig

Dim_Investigations

As shown in Figure 97, the Dim_Investigations package uses the stored procedure usp_DWInvestigationDetail to update the Dim_Investigations table and load the DCDW data warehouse on the REDACTED server. This stored procedure extracts all the investigations that occur in facilities at the inmate level from the REDACTED server in the IGLOGS database. The Dim_Investigations table is separated by inmate, because more than one record can exist with the same investigation log number when more than one inmate is involved. In addition, the Dim_Investigations package adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTED server.

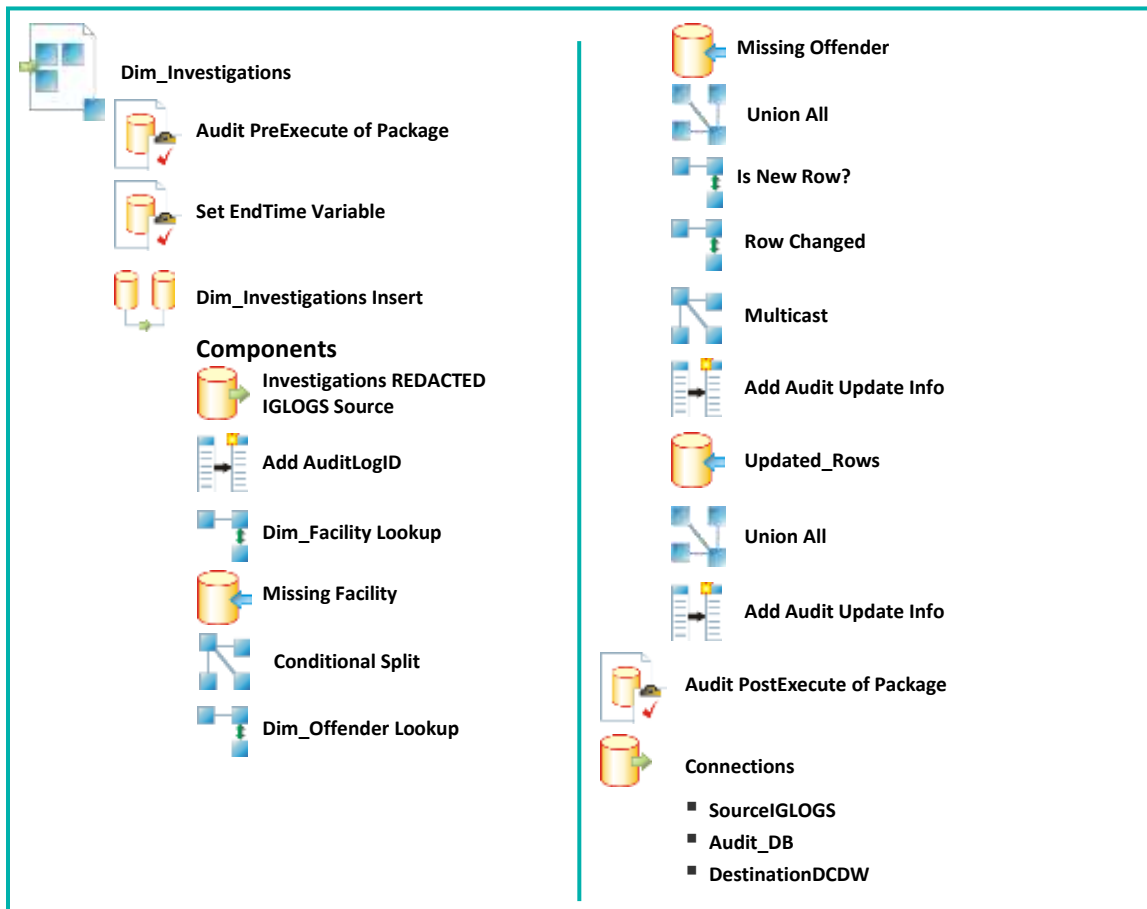


Figure 97: Contents of the Dim_Investigations package

Audit PreExecute of Package

As shown in Figure 98, the Audit PreExecute of Package execute SQL task uses the stored procedure `up_Event_Package_OnBegin` to add auditing data to the table `ExecutionLog` in the `Audit_DB`. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

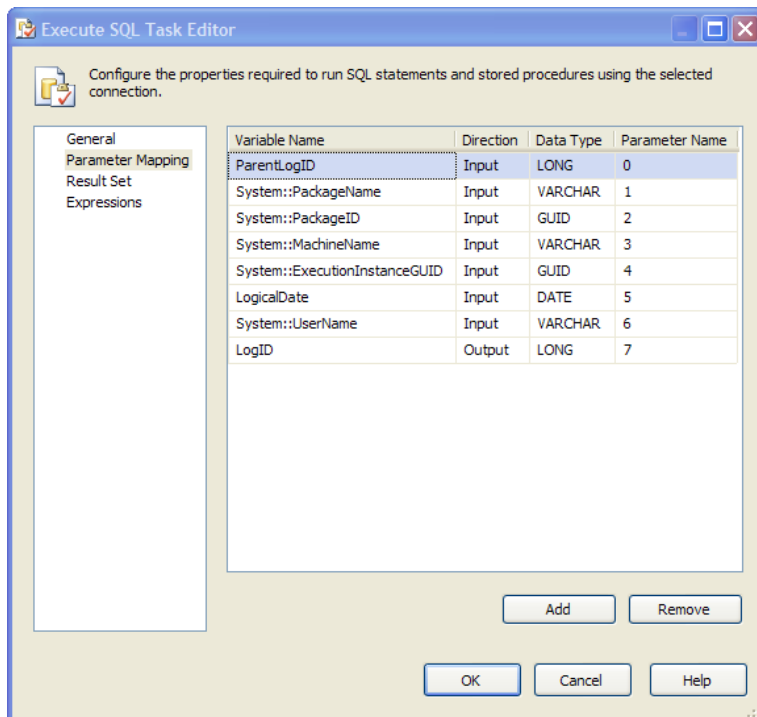


Figure 98: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_Investigations Inserts

As shown in Figure 99, the Dim_Investigations Inserts data flow task uses the usp_DWInvestigationDetail stored procedure from the IGLOGS database on the REDACTED server to update the Dim_Investigations table in the data warehouse.

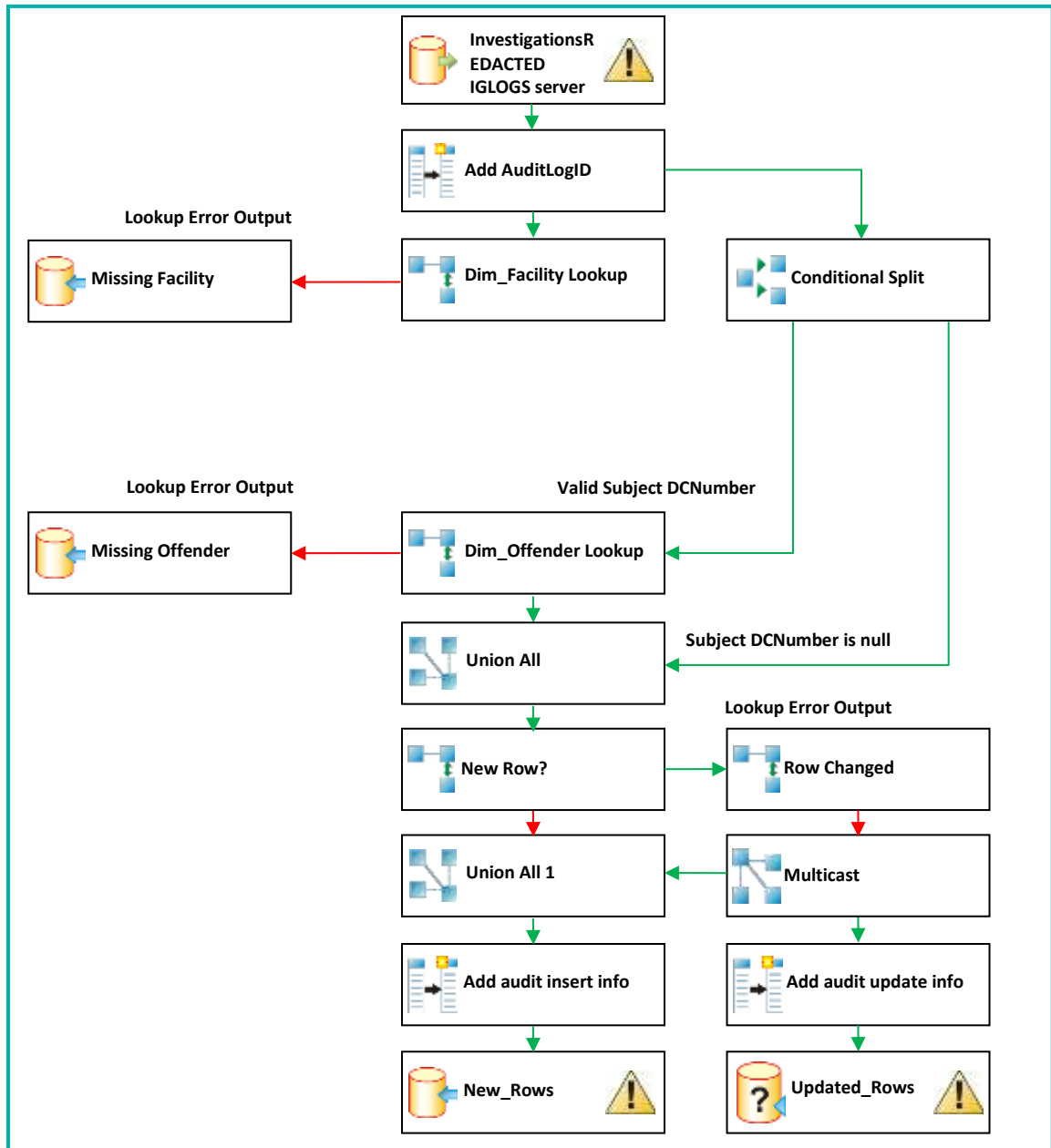


Figure 99: Dim_Investigations Inserts data flow task

Table 35 describes the components of the Dim_Investigations Inserts data flow task.

Table 35: Components of the Dim_Investigations Inserts data flow task		
Symbol	Data flow component	Description
	Investigations REDACTED IGLOGS Source	Brings data in from source REDACTED IGLOGS) stored procedure usp_DWInvestigationDetail
	Add AuditLogID	Adds column AuditLogID to identify the package with a run number
	Dim_Facility Lookup	Lookup that checks and matches source facilities with those already in the Dim_Facility data warehouse table
	Missing Facility	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIds)
	Conditional Split	Conditional Split that separates SubjectDCNumber before pref Dim_Offender Lookup
	Dim_Offender Lookup	Lookup that checks and matches source offenders with those already in the data warehouse table Dim_Offender
	Missing Offender	Places offenders that do not match Dim_Offender table in the Missing Offender Keys audit table (Brings together null SubjectDCNumber and valid SubjectDCNumber.)
	Union All	Joins newly null SubjectDCNumbers and valid SubjectDCNumbers that have been modified
	Is New Row?	This Lookup transform finds out if any rows that are loaded are newly entered
	Row Changed	Lookup checks to see if the row is not new has it been modified
	Multicast	Sends the results of the lookup to update command and Dim_Investigations table
	Add Audit Update Info	Updated rows have columns RowEndDate and UpdateAudit_ID added
	Updated_Rows	OLE DB Command updates rows with new data
	Union All 1	Brings together New and Updated Rows
	Add Audit Insert Info	New rows have columns RowStartDate and InsertAudit_ID added
	New_Rows	New Rows are loaded into Dim_Investigations in data warehouse

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 36 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 36: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceIGLOGS		REDACTED	IGLOGS	REDACTED\config\SQL\SSIS\SourceIGLOGS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

GrievanceOfficeDetail

As shown in Figure 100, the GrievanceOfficeDetail package uses the stored procedure usp_DWOfficeGrievanceDetail to update the GrievanceOfficeDetail table to load the DCDW data warehouse on the REDACTEDserver. In addition, this stored procedure extracts all the grievances filed to the Central Office that occur in facilities gathered from the server REDACTEDin the IGLOGS database. Finally, this package adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTEDserver.

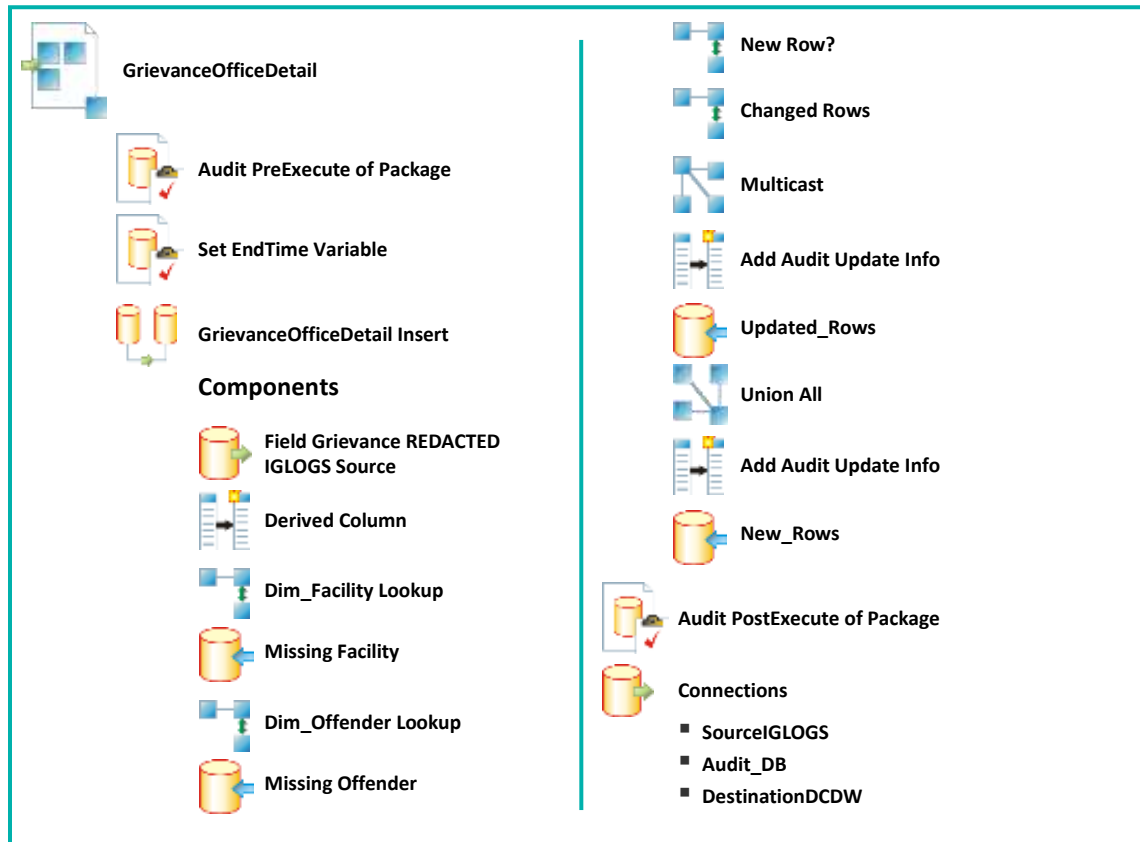


Figure 100: Contents of the GrievanceOfficeDetail package

Audit PreExecute of Package

As shown in Figure 101, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

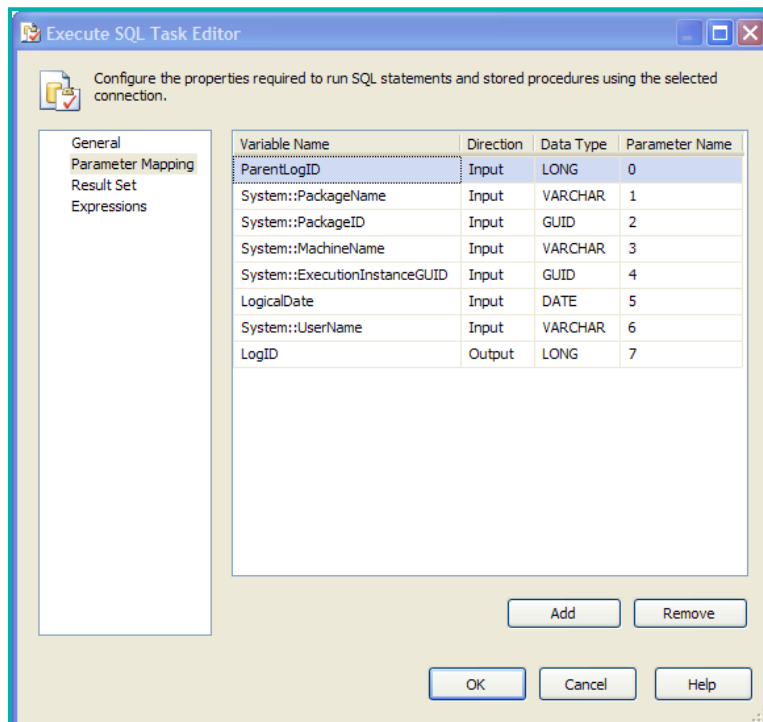


Figure 101: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

GrievanceOfficeDetail Insert

As shown in Figure 102, the GrievanceOfficeDetail Insert data flow task the usp_DWOfficeGrievanceDetail stored procedure from the IGLOGS database on the REDACTEDserver to update the GrievanceOfficeDetail table in the data warehouse.

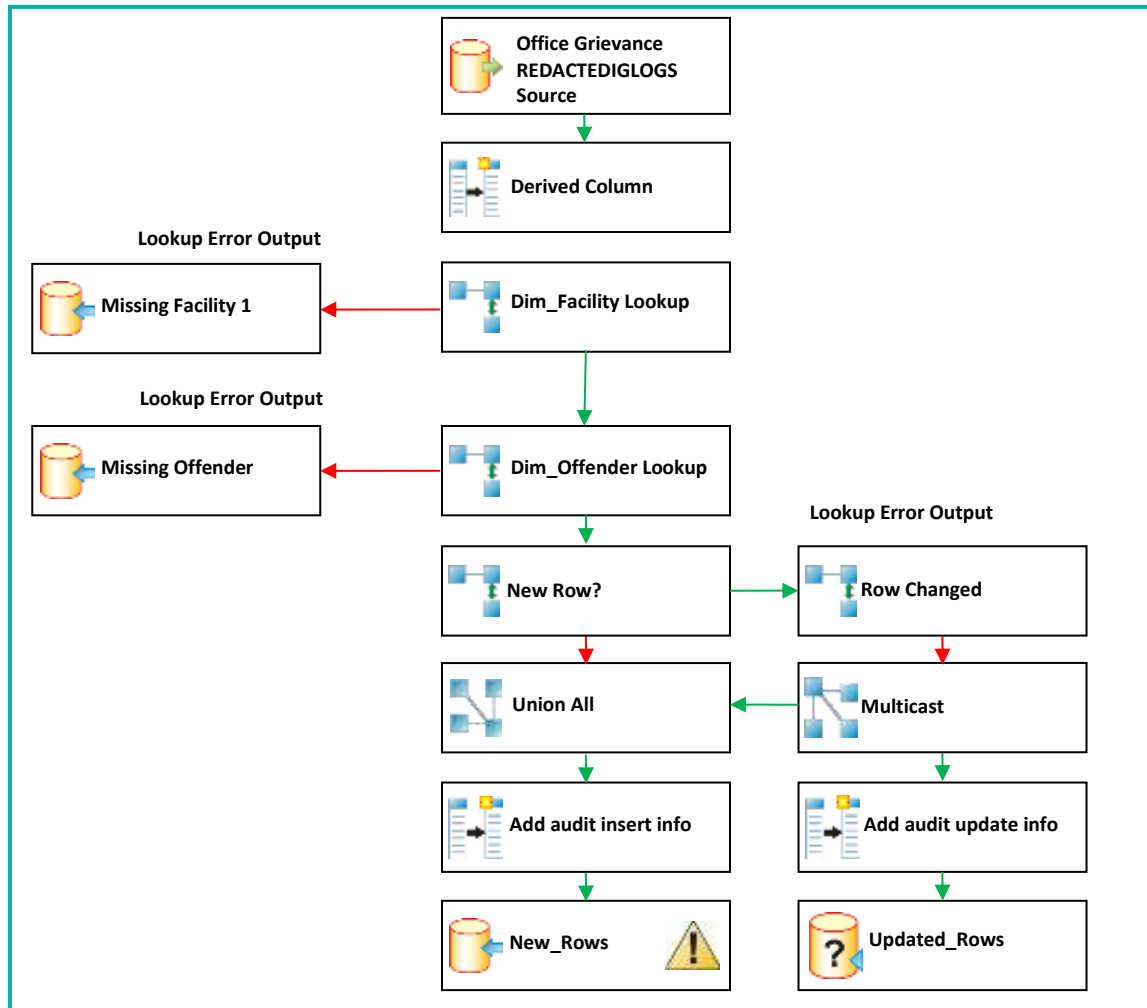


Figure 102: GrievanceOfficeDetail Insert data flow task

Table 37 describes the components of the GrievanceOfficeDetail Insert data flow task.

Table 37: Components of the GrievanceOfficeDetail Insert data flow task		
Symbol	Data flow component	Description
	Office Grievance REDACTED IGLOGS Source	Brings in data from source REDACTED IGLOGS) stored procedure usp_DWOfficeGrievanceDetail
	Derived Column	Adds column AuditLogID to identify the package with a run number

Table 37: Components of the GrievanceOfficeDetail Insert data flow task

Symbol	Data flow component	Description
	Dim_Facility Lookup	Lookup that checks and matches source facilities with those already in the Dim_Facility table of the data warehouse
	Missing Facility	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIDs)
	Dim_Offender Lookup	Lookup that checks and matches source offenders with those already in the data warehouse table Dim_Offender
	Missing Offender	Places offenders that do not match Dim_Offender table in an audit table (Missing Offender Keys)
	New Row?	This lookup transform finds out if any rows that are loaded are newly entered
	Changed Rows	This lookup checks to see if the row has been modified if it is not new
	Union All	Brings together new and updated rows
	Multicast	Sends the results of the lookup to update command and GrievanceOfficeDetail table
	Add Audit Insert Info	New rows have columns RowStartDate and InsertAudit_ID added
	Add Audit Update Info	Updated rows have columns RowEndDate and UpdateAudit_ID added
	New_Rows	New rows are loaded into GrievanceOfficeDetail in the data warehouse
	Updated_Rows	OLE DB command updates rows with new data

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 38 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 38: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
	SourceIGLOGS	REDACTED	IGLOGS	\\REDACTED\\config\\SQL\\SSIS\\SourceIGLOGS.dtsConfig
	Audit_DB	REDACTED	Audit_DB	\\REDACTED\\config\\SQL\\SSIS\\AuditDestination.dtsConfig
	DestinationDCDW	REDACTED	DCDW	\\REDACTED\\config\\SQL\\SSIS\\DestinationDCDW.dtsConfig

GrievanceFieldDetail

As shown in Figure 103, the GrievanceFieldDetail packages uses the stored procedure usp_DWFieldGrievanceDetail to update the GrievanceFieldDetail table and load the DCDW data warehouse on the REDACTED server. This stored procedure extracts all the grievances filed out in the field or facility gathered from the server REDACTED in the IGLOGS database. It also adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTED server.

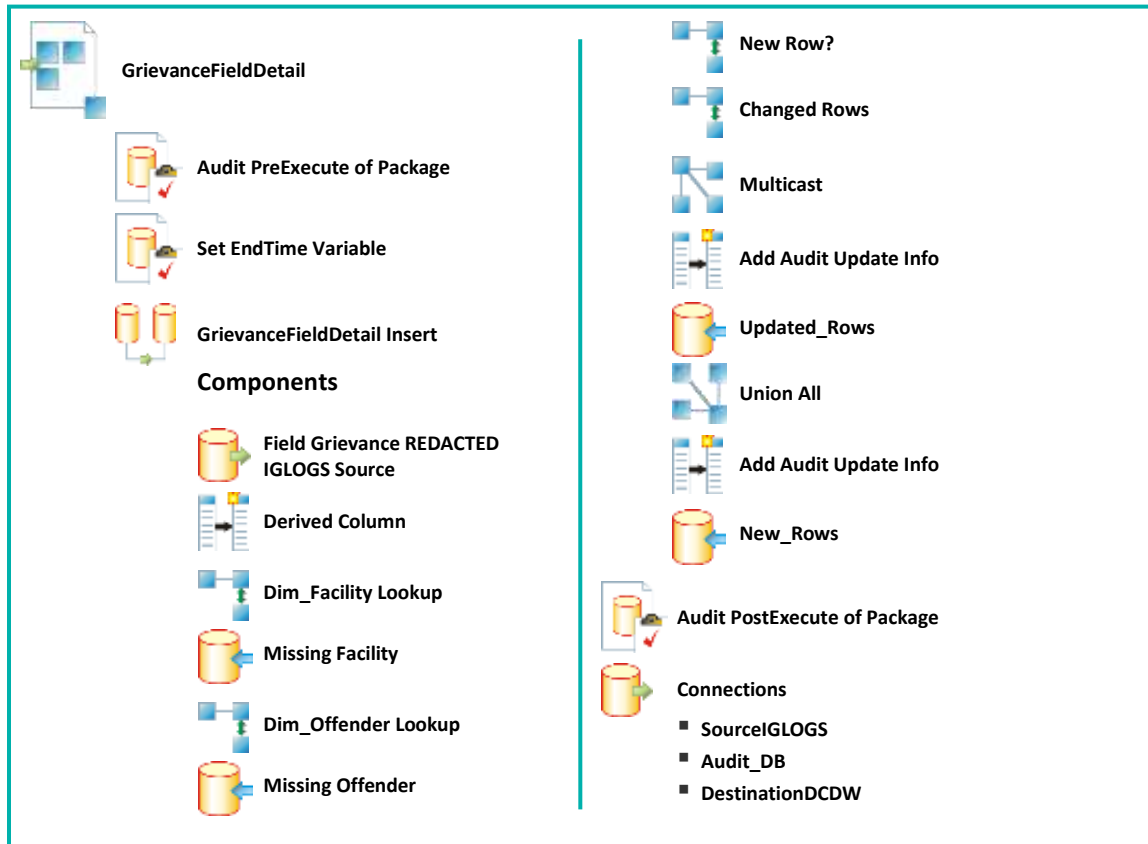


Figure 103: Contents of the GrievanceFieldDetail package

Audit PreExecute of Package

As shown in Figure 104, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

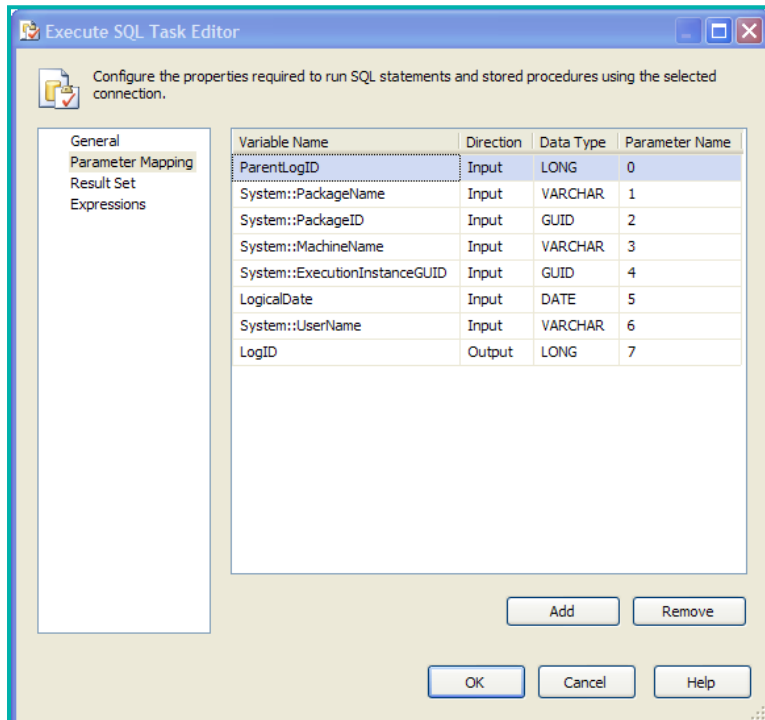


Figure 104: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

GrievanceFieldDetail Insert

As shown in Figure 105, the GrievanceFieldDetail Insert data flow task uses the usp_DWOfficeGrievanceDetail stored procedure from IGLOGS on the REDACTED server to update the GrievanceFieldDetail table in the data warehouse.

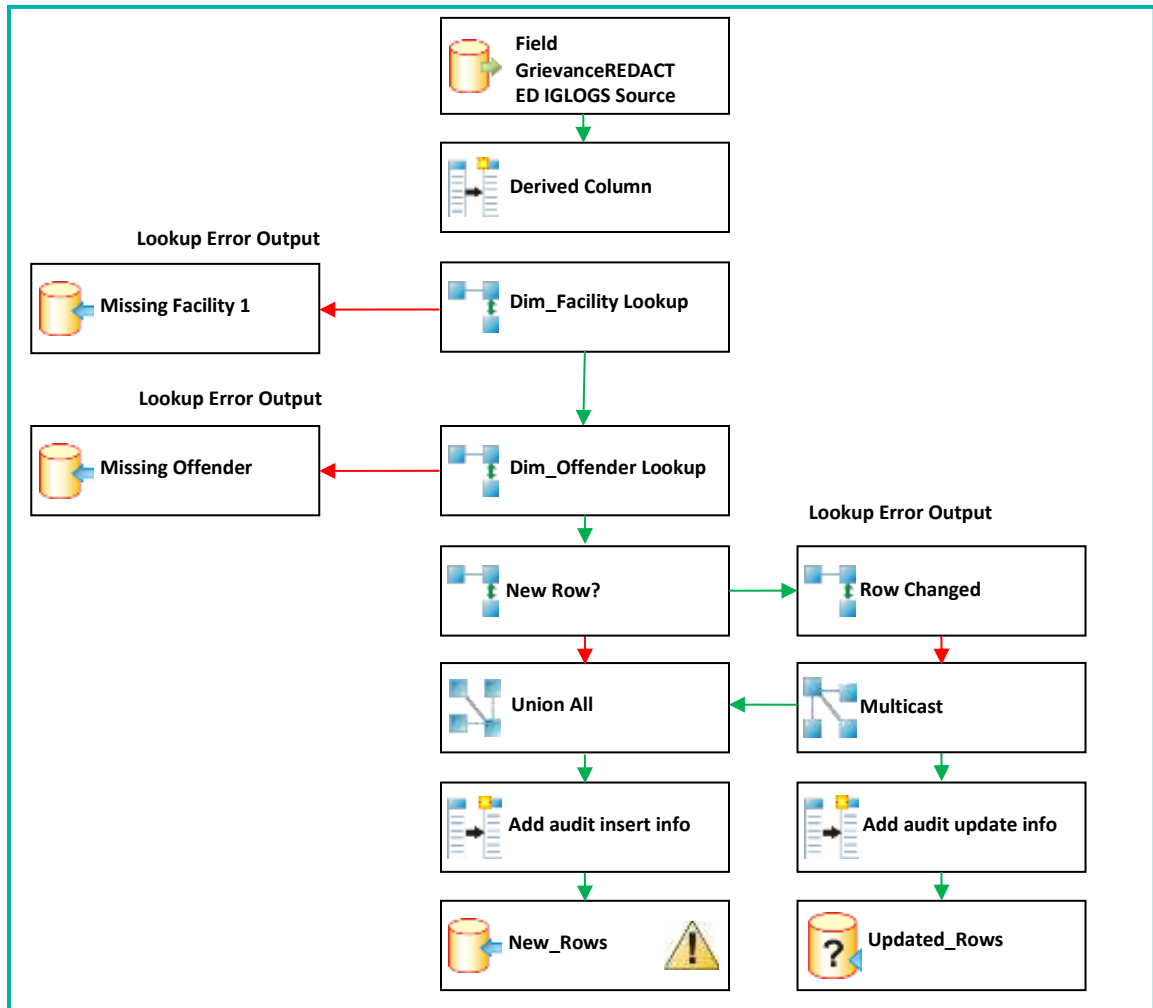
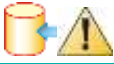


Figure 105: GrievanceFieldDetail Insert data flow task

Table 39 describes the components of the GrievanceFieldDetail Insert data flow task.

Table 39: Components of the GrievanceFieldDetail Insert data flow task		
Symbol	Data flow component	Description
	Field GrievanceREDACTEDIGLOGS Source	Brings in data from source (REDACTEDIGLOGS) stored procedure usp_DWOfficeGrievanceDetail
	Derived Column	Adds column AuditLogID to identify the package with a run number

Table 39: Components of the GrievanceFieldDetail Insert data flow task

Symbol	Data flow component	Description
	Dim_Facility Lookup	Lookup that checks and matches source facilities with those already in the Dim_Facility table of the data warehouse
	Missing Facility	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIDs)
	Dim_Offender Lookup	Lookup that checks and matches source offenders with those already in the data warehouse table Dim_Offender
	Missing Offender	Places offenders that do not match Dim_Offender table in an audit table (Missing Offender Keys)
	New Row?	This lookup transform finds out if any rows that are loaded are newly entered
	Changed Rows	This lookup checks to see if a row has been modified if it is not new
	Union All	Brings together new and updated rows
	Multicast	Sends the results of the lookup to update command and GrievanceFieldDetail table
	Add Audit Insert Info	New rows have columns RowStartDate and InsertAudit_ID added
	Add Audit Update Info	Updated rows have columns RowEndDate and UpdateAudit_ID added
	New_Rows	New rows are loaded into GrievanceFieldDetail in the data warehouse
	Updated_Rows	OLE DB command updates rows with new data

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure `audit.up_Event_Package_OnEnd` to add auditing data to the table `ExecutionLog` in the `Audit_DB`. The purpose of this task is to complete the audit table by adding the time the package completed in the column `EndTime`. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 40 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 40: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceIGLOGS		REDACTED	IGLOGS	REDACTED\config\SQL\SSIS\SourceIGLOGS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

Dim_DisciplinaryReports

As shown in Figure 106, the `Dim_DisciplinaryReports` package uses the stored procedure `usp_Dim_DisciplinaryReports` to update the `Dim_DisciplinaryReports` table and load the DCDW data warehouse on the REDACTED server. This stored procedure extracts all the disciplinary reports gathered from the REDACTED server in the DCIS database and adds auditing data to the `ExecutionLog` table of the `Audit_DB` database on the REDACTED server.

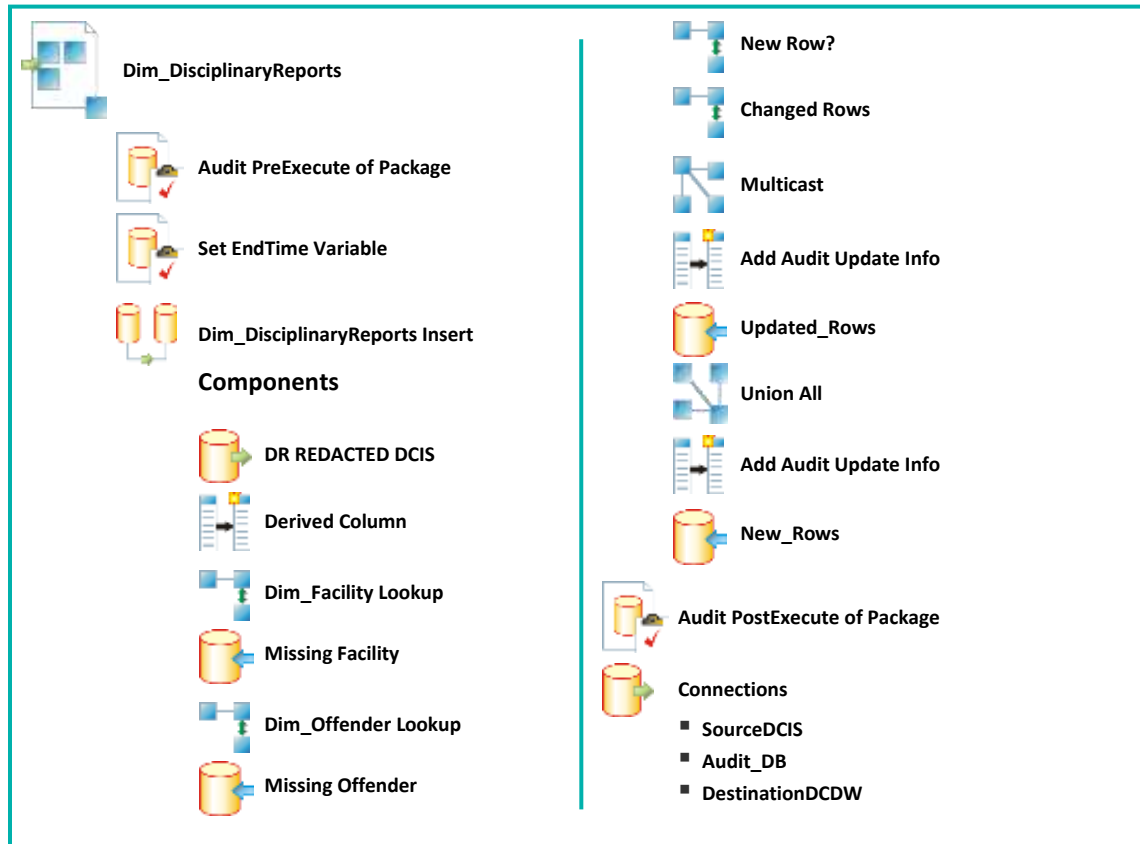


Figure 106: Contents of the Dim_DisciplinaryReports package

Audit PreExecute of Package

As shown in Figure 107, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

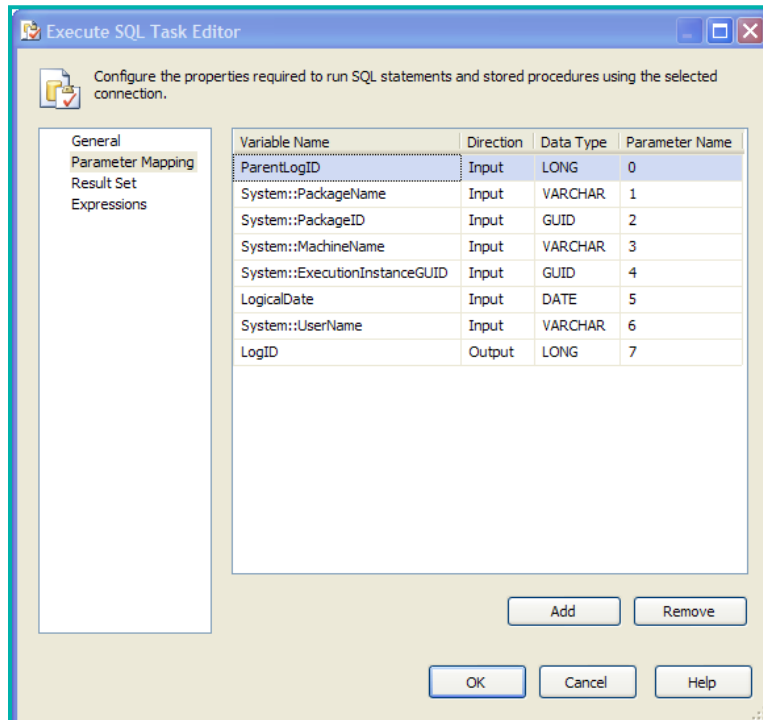


Figure 107: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_DisciplinaryReports Insert

As shown in Figure 108, the Dim_DisciplinaryReports data flow task uses the usp_Dim_DisciplinaryReports stored procedure from DCIS on the REDACTED server to update the Dim_DisciplinaryReports table in the data warehouse.

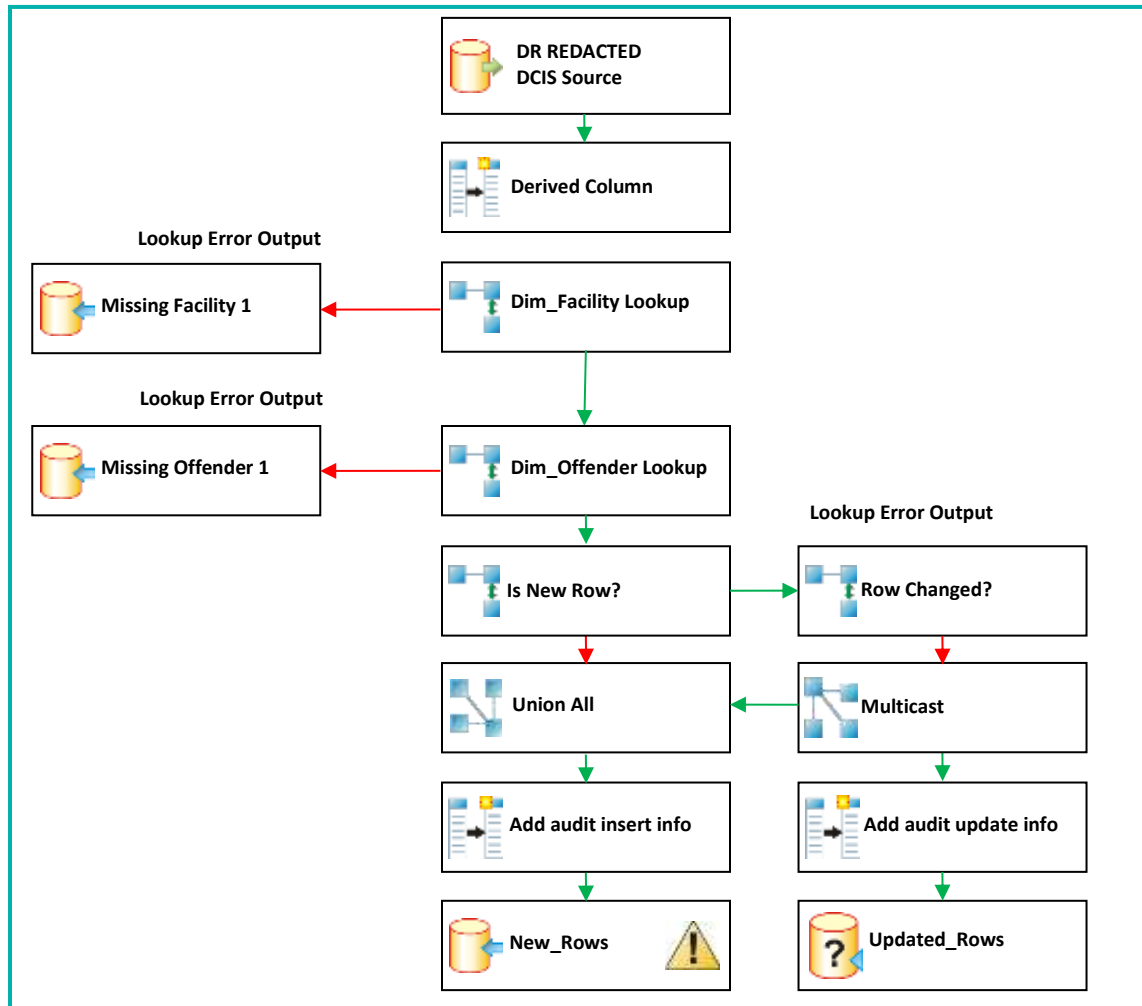


Figure 108: GrievanceFieldDetail Insert data flow task

Table 41 describes the components of the Dim_DisciplinaryReports Insert data flow task.

Table 41: Components of the Dim_ManagementScores_Offender Inserts data flow task		
Symbol	Data flow component	Description
	DR REDACTED DCIS Source	Brings in data from source (DCIS) stored procedure usp_Dim_DisciplinaryReports
	Derived Column	Adds column AuditLogID to identify the package with a run number
	Dim_Facility Lookup	Lookup that checks and matches source facilities with those already in the Dim_Facility table of the data warehouse

Table 41: Components of the Dim_ManagementScores_Offender Inserts data flow task

Symbol	Data flow component	Description
	Missing Facility	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIDs)
	Dim_Offender Lookup	Lookup that checks and matches source offenders with those already in the data warehouse table Dim_Offender
	Missing Offender	Places offenders that do not match Dim_Offender table in an audit table (Missing Offender Keys)
	New Row?	This lookup transform finds out if any rows that are loaded are newly entered
	Changed Rows	This lookup checks to see if a row has been modified if it is not new
	Union All	Brings together new and updated rows
	Multicast	Sends the results of the lookup to update command and Dim_DisciplinaryReports table
	Add Audit Insert Info	New rows have columns RowStartDate and InsertAudit_ID added
	Add Audit Update Info	Updated rows have columns RowEndDate and UpdateAudit_ID added
	New_Rows	New rows are loaded into Dim_DisciplinaryReports in the data warehouse
	Updated_Rows	OLE DB command updates rows with new data

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 42 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 42: Connections for the Audit PostExecute of Package task

Connection	Target server	Target database	Configuration file
 SourceDCIS	REDACTED	DCIS	REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig

Table 42: Connections for the Audit PostExecute of Package task

Connection	Target server	Target database	Configuration file
 Audit_DB	REDACTED	Audit_DB	REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW	REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

UOFDetail

As shown in Figure 109, the UOFDetail package updates the table UOFDetail using the stored procedure usp_DWUOF to load the data warehouse database DCDW located on the REDACTEDserver. This stored procedure extracts all the use of force reports at the inmate level gathered from the server REDACTEDin the IGLOGS database. It is important to remember that more than one inmate may be involved with the use-of-force event. Consequently, this table is separated by inmate and more than one record may exist with the same use-of-force (UOF) log number. Finally, this table adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTEDserver.

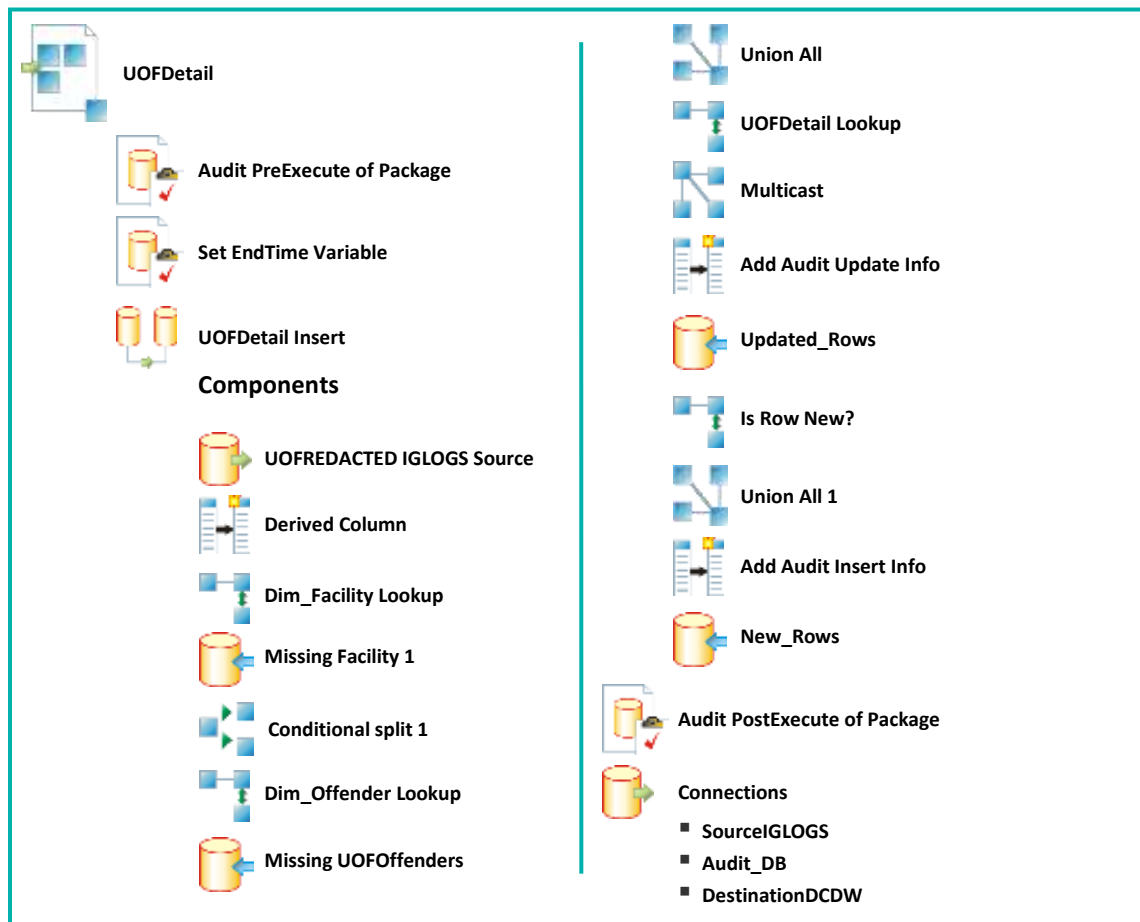


Figure 109: Contents of the UOFDetail package

Audit PreExecute of Package

As shown in Figure 110, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

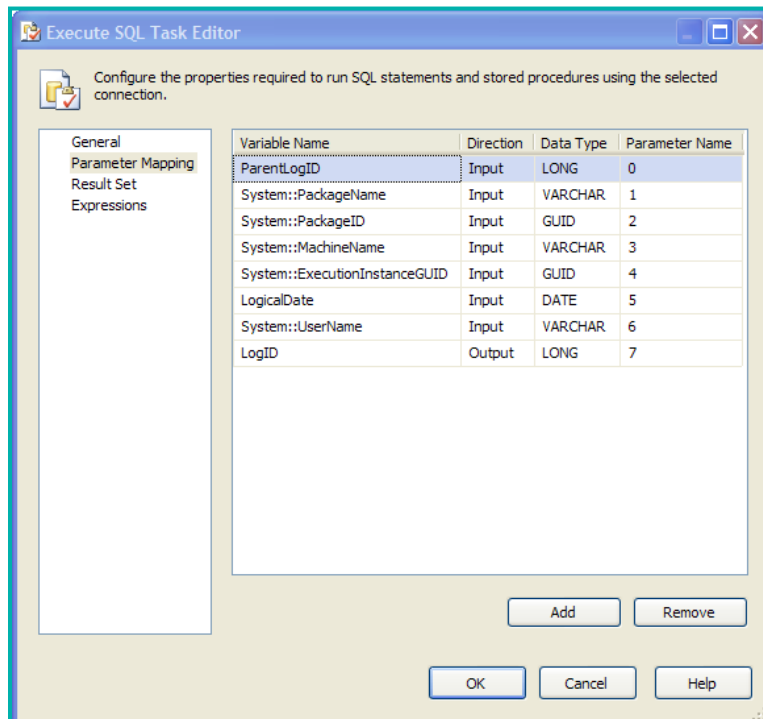


Figure 110: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

UOFDetail Insert

As shown in Figure 111, the UOFDetail data flow task uses the usp_DWUOF stored procedure from the IGLOGS database on the REDACTED server to update the UOFDetail table in the data warehouse.

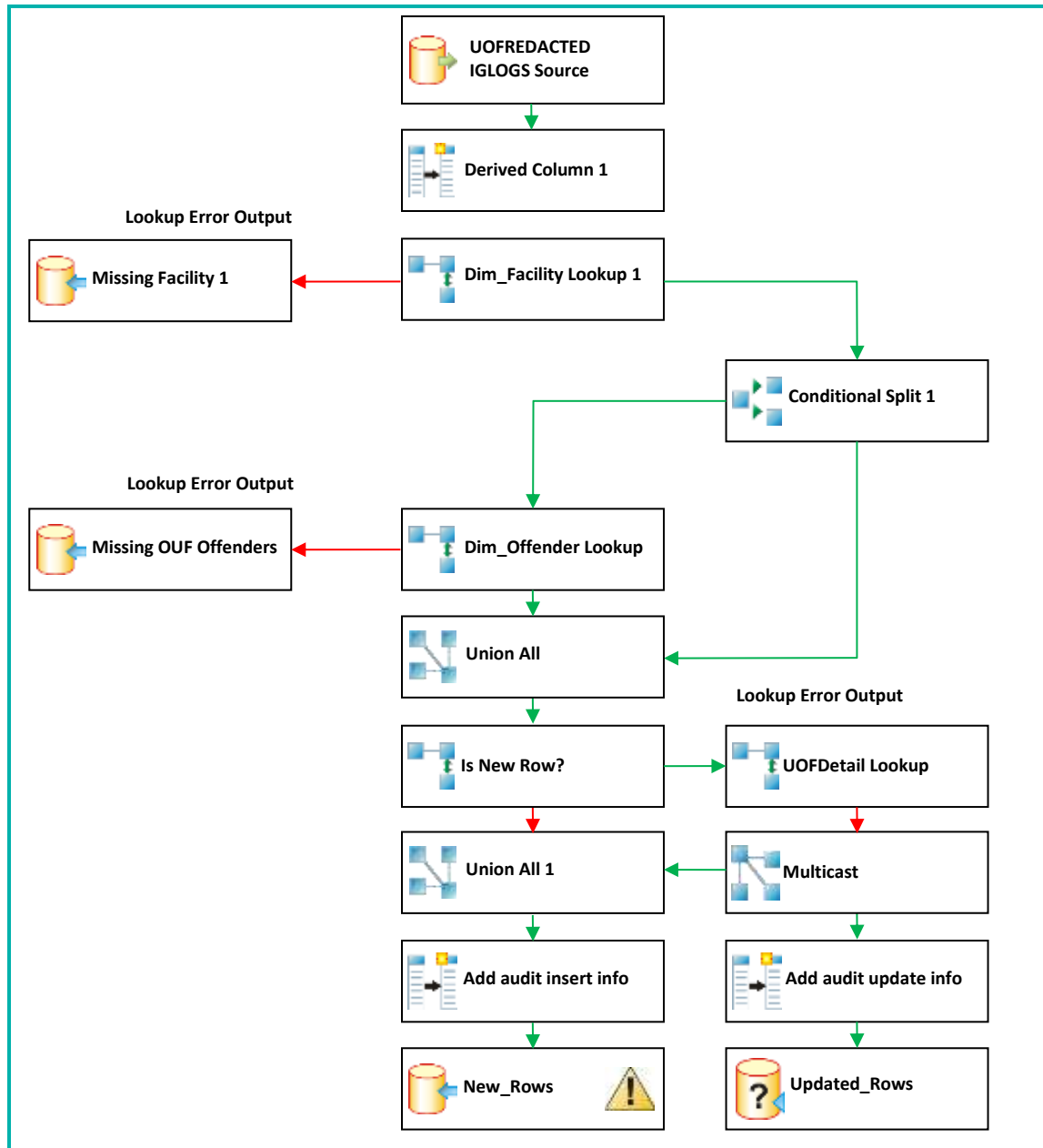


Figure 111: UOFDetail Insert data flow task

Table 43 describes the components of the UOFDetail Insert data flow task.

Table 43: Components of the UOFDetail Inserts data flow task		
Symbol	Data flow component	Description
	UOF REDACTED IGLOGS Source	Brings in data from source (REDACTED IGLOGS) stored procedure usp_DWUOF
	Derived Column 1	Adds column AuditLogID to identify the package with a run number
	Dim_Facility Lookup 1	Lookup that checks and matches source facilities with those already in the Dim_Facility table of the data warehouse
	Missing Facility 1	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIDs)
	Dim_Offender Lookup	Lookup that checks and matches source offenders with those already in the data warehouse table Dim_Offender
	Conditional Split 1	Conditional split that tests to see SubjectDCNumber Exists
	Missing Offender	Places offenders that do not match Dim_Offender table in an audit table (Missing Offender Keys)
	Union All	Joins newly null SubjectDCNumbers and valid SubjectDCNumbers that have been modified
	Is Row New?	This lookup transform finds out if any rows that are loaded are newly entered
	UOFDetail Lookup	This lookup checks to see if a row has been modified if it is not new
	Union All 1	Brings together new and updated rows
	Multicast	Sends the results of the lookup to update command and UOFDetail table
	Add Audit Insert Info	New rows have columns RowStartDate and InsertAudit_ID added
	Add Audit Update Info	Updated rows have columns RowEndDate and UpdateAudit_ID added
	New_Rows	New Rows are loaded into UOFDetail in data warehouse
	Updated_Rows	OLE DB command updates rows with new data

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure `audit.up_Event_Package_OnEnd` to add auditing data to the table `ExecutionLog` in the `Audit_DB`. The purpose of this task is to complete the audit table by adding the time the package completed in the column `EndTime`. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 44 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 44: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceIGLOGS		REDACTED	IGLOGS	REDACTED\config\SQL\SSIS\SourceIGLOGS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	REDACTEDconfig\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

Dim_Escape

As shown in Figure 112, Dim_Escape uses the usp_Dim_Escape stored procedure to update the Dim_Escape table and load the DCDW data warehouse on the REDACTEDserver. This stored procedure extracts all the escapes gathered from the server REDACTEDin the DCIS database and adds auditing data to the ExecutionLog table of the Audit_DB database on the REDACTEDserver.

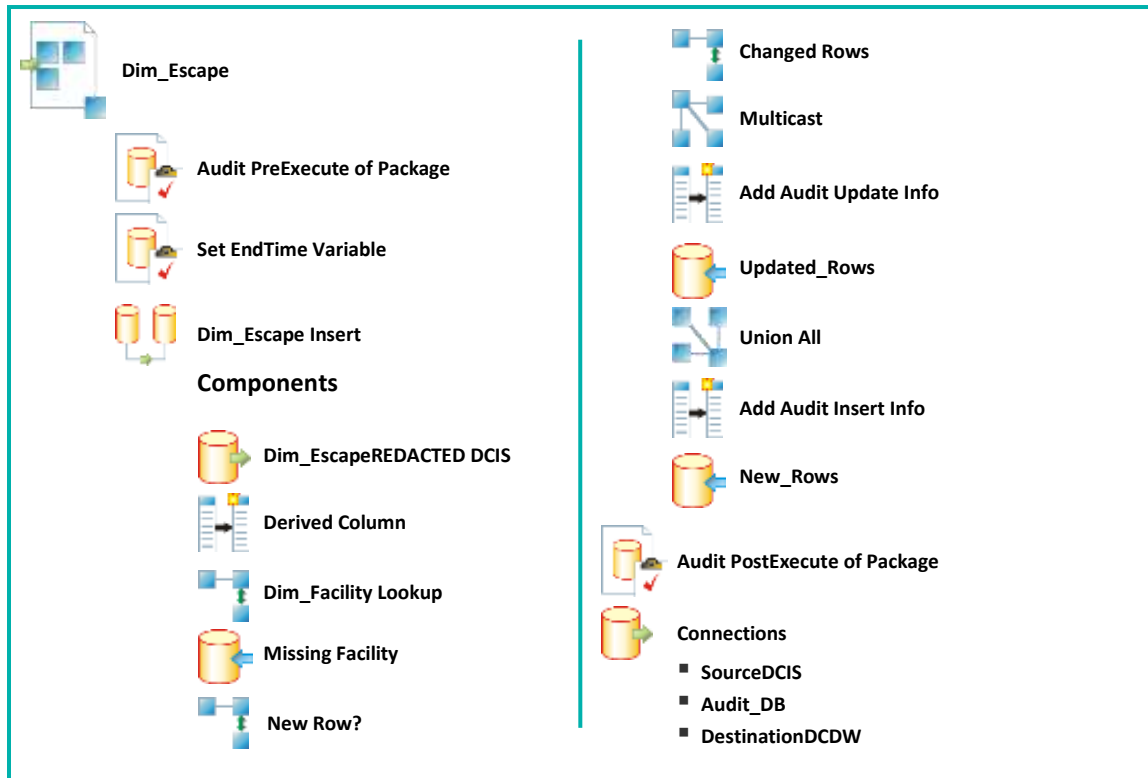


Figure 112: Contents of the Dim_Escape package

Audit PreExecute of Package

As shown in Figure 113, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

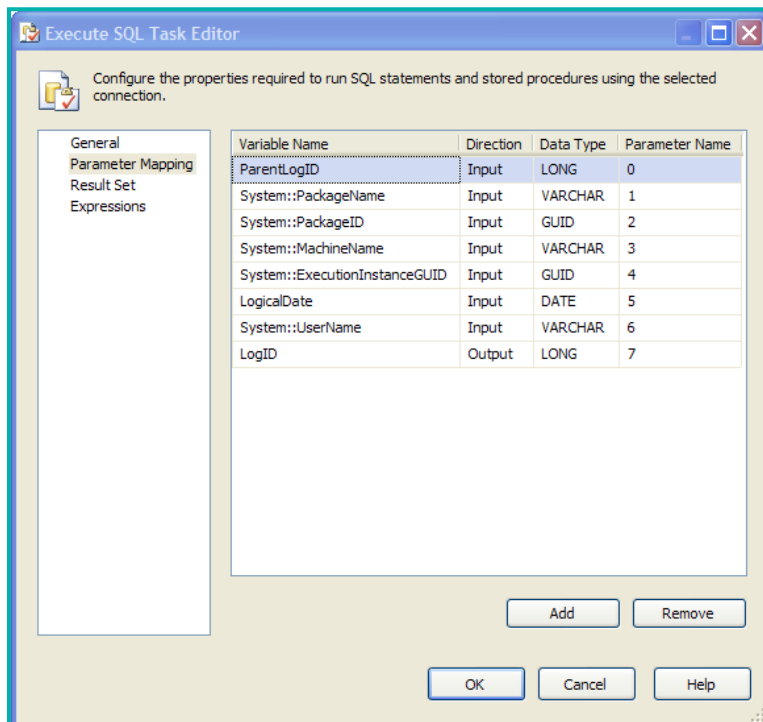


Figure 113: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
exec audit.up_Update_Time_Set ?
```

Dim_Escape Insert

As shown in Figure 114, the Dim_Escape data flow task uses the usp_Dim_Escape stored procedure from the IGLOGS database on the EDACTED server to update the Dim_Escape table in the data warehouse.

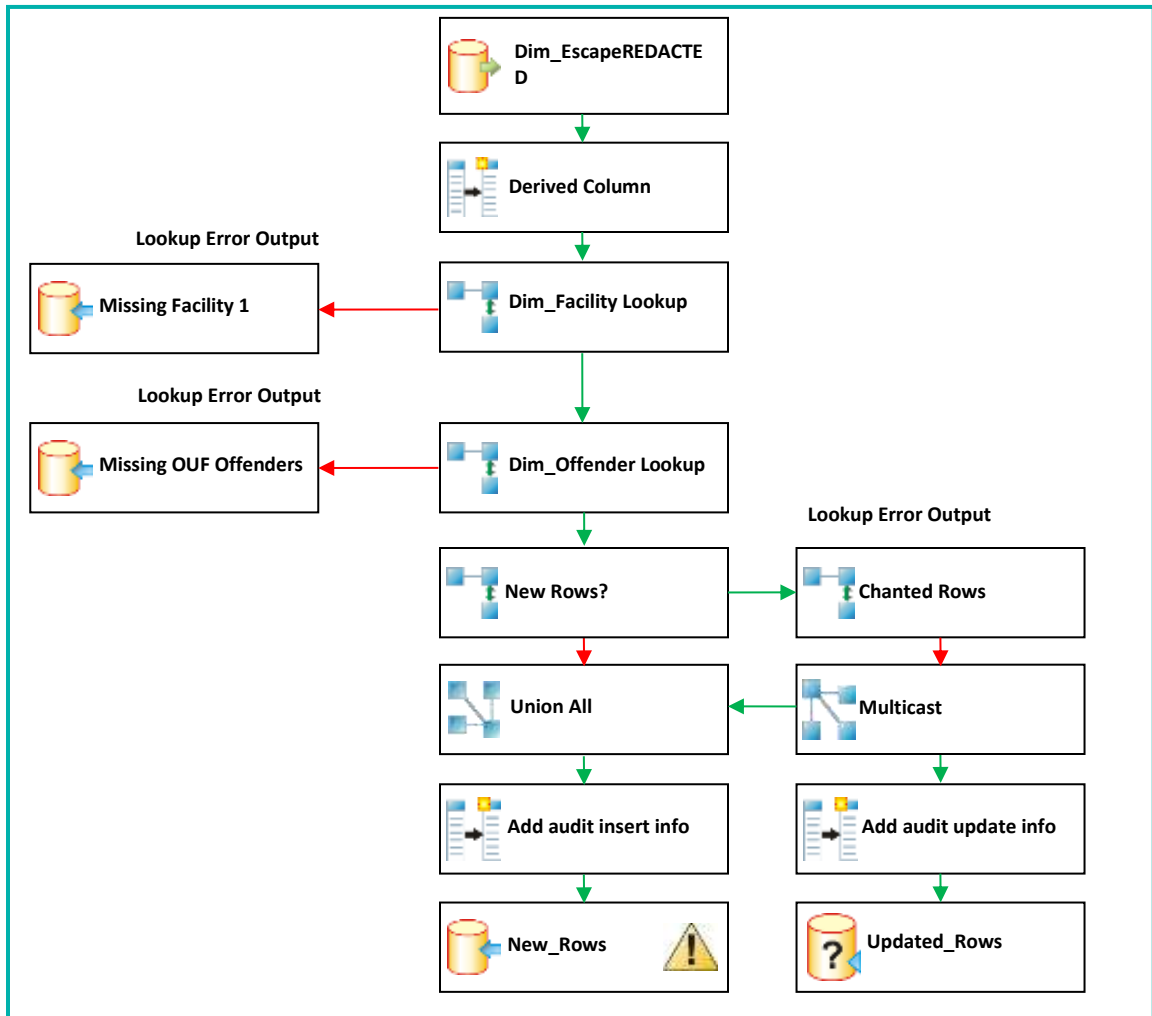


Figure 114: Dim_Escape Insert data flow task

Table 45 describes the components of the Dim_Escape Insert data flow task.

Table 45: Components of the Dim_Escape Insert data flow task		
Symbol	Data flow component	Description
	UOF REDACTED IGLOGS Source	Brings in data from source (REDACTED IGLOGS) stored procedure usp_DWUOF
	Derived Column 1	Adds column AuditLogID to identify the package with a run number

Table 45: Components of the Dim_Escape Insert data flow task

Symbol	Data flow component	Description
	Dim_Facility Lookup	Lookup that checks and matches source facilities with those already in the Dim_Facility table of the data warehouse
	Missing Facility 1	Places facilities that do not match Dim_Facility table in an audit table (MissingFacilityIDs)
	Dim_Offender Lookup	Lookup that checks and matches source offenders with those already in the data warehouse table Dim_Offender
	Conditional Split 1	Conditional split that tests to see SubjectDCNumber Exists
	Missing Offender	Places offenders that do not match Dim_Offender table in an audit table (Missing Offender Keys)
	Row New?	This lookup transform finds out if any rows that are loaded are newly entered
	Changed Rows	This lookup checks to see if a row has been modified if it is not new
	Union All	Brings together new and updated rows
	Multicast	Sends the results of the lookup to update command and Dim_Escape table
	Add Audit Insert Info	New rows have columns RowStartDate and InsertAudit_ID added
	Add Audit Update Info	Updated rows have columns RowEndDate and UpdateAudit_ID added
	New_Rows	New Rows are loaded into Dim_Escape in data warehouse
	Updated_Rows	OLE DB command updates rows with new data

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure `audit.up_Event_Package_OnEnd` to add auditing data to the table `ExecutionLog` in the `Audit_DB`. The purpose of this task is to complete the audit table by adding the time the package completed in the column `EndTime`. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 46 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 46: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceDCIS		REDACTED	DCIS	\\REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	\\REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	\\REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

Fact_Event

As shown in Figure 115, the `Fact_Event` package uses the `V_FactEvent_All` view to update the `Fact_Event` table and load the DCDW data warehouse on the REDACTED server. In addition, this view compiles all the events from the prior dimension tables gathered from DCDW. Finally, it adds auditing data to the `ExecutionLog` table of the `Audit_DB` database on the REDACTED server.

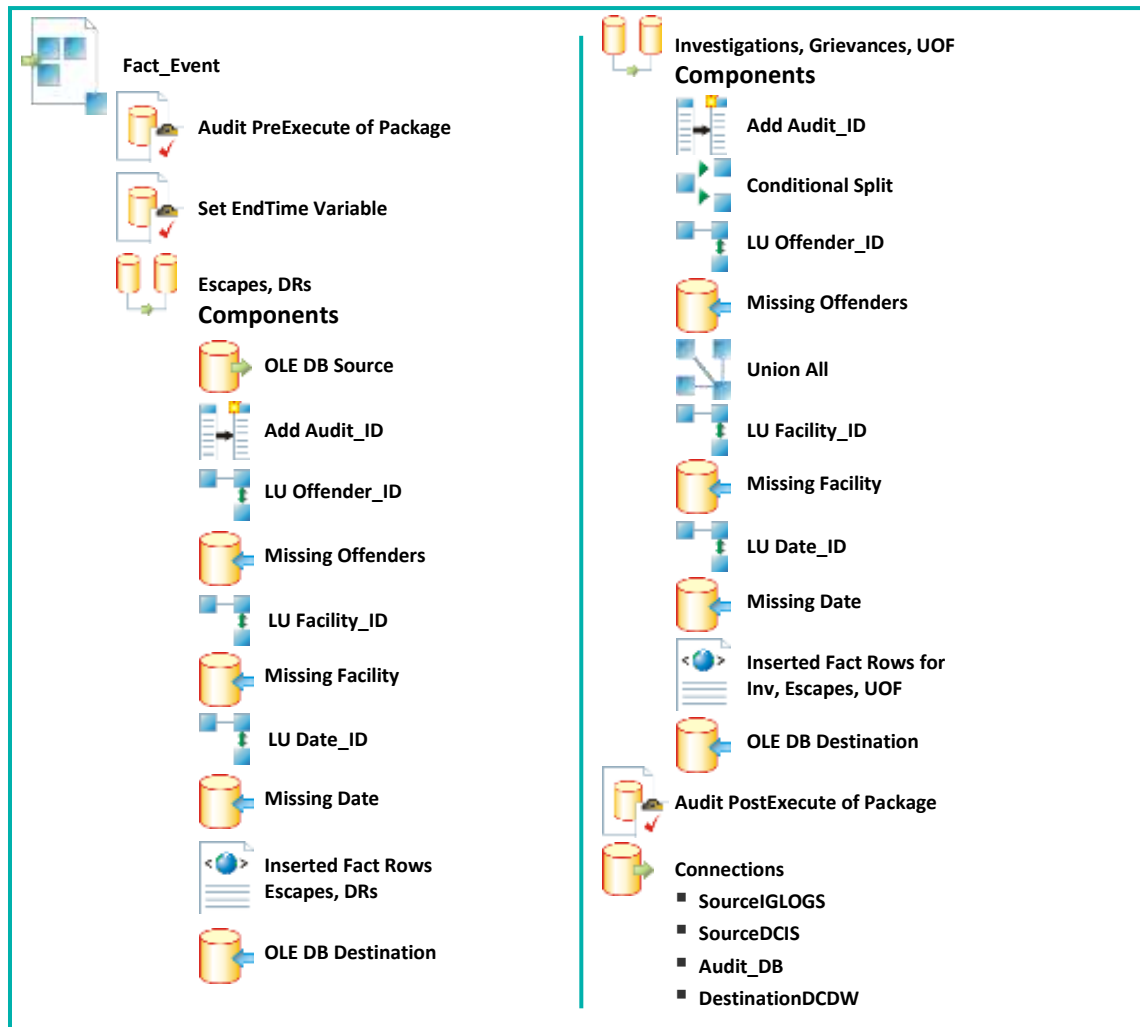


Figure 115: Contents of the Fact_Event package

Audit PreExecute of Package

As shown in Figure 116, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

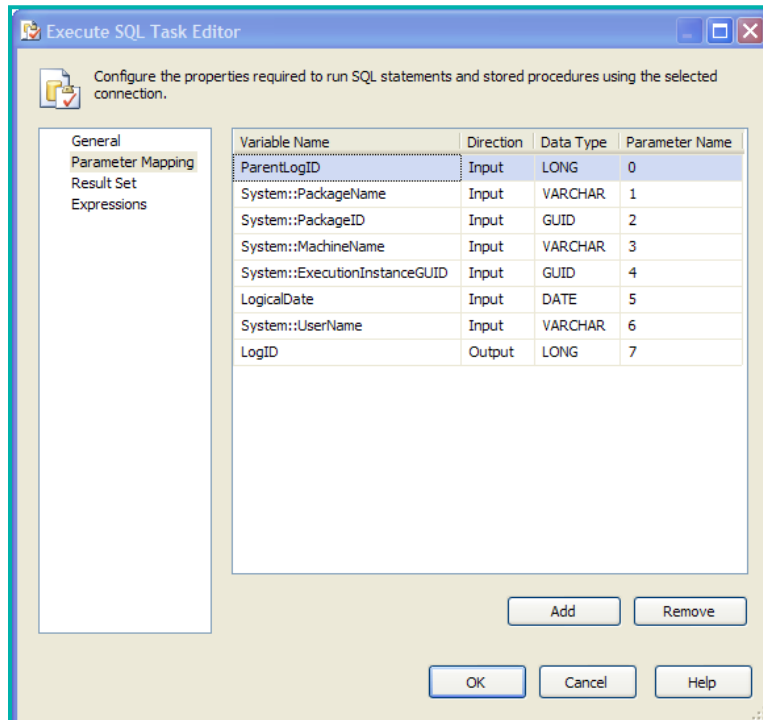


Figure 116: Execute SQL Task Editor Dialog box for the Audit PreExecute of Package execute SQL task

Set EndTime Variable

By running the following SQL statement, Set EndTime Variable sets the EndTime variable that later assists with returning only the rows that have not been loaded yet:

```
SELECT max(date_ID+time_id)As EndTime
FROM Fact_Event
```

Escapes, DRs

As shown in Figure 117, the Escapes and DRs data flow task uses the usp_FACT_Event stored procedure from the DCIS database on the REDACTED server to create the FACT_EVENT table in the data warehouse. This task also handles data errors in Offender_ID, Date_ID and Facility_ID by placing them in the Audit_DB database tables.

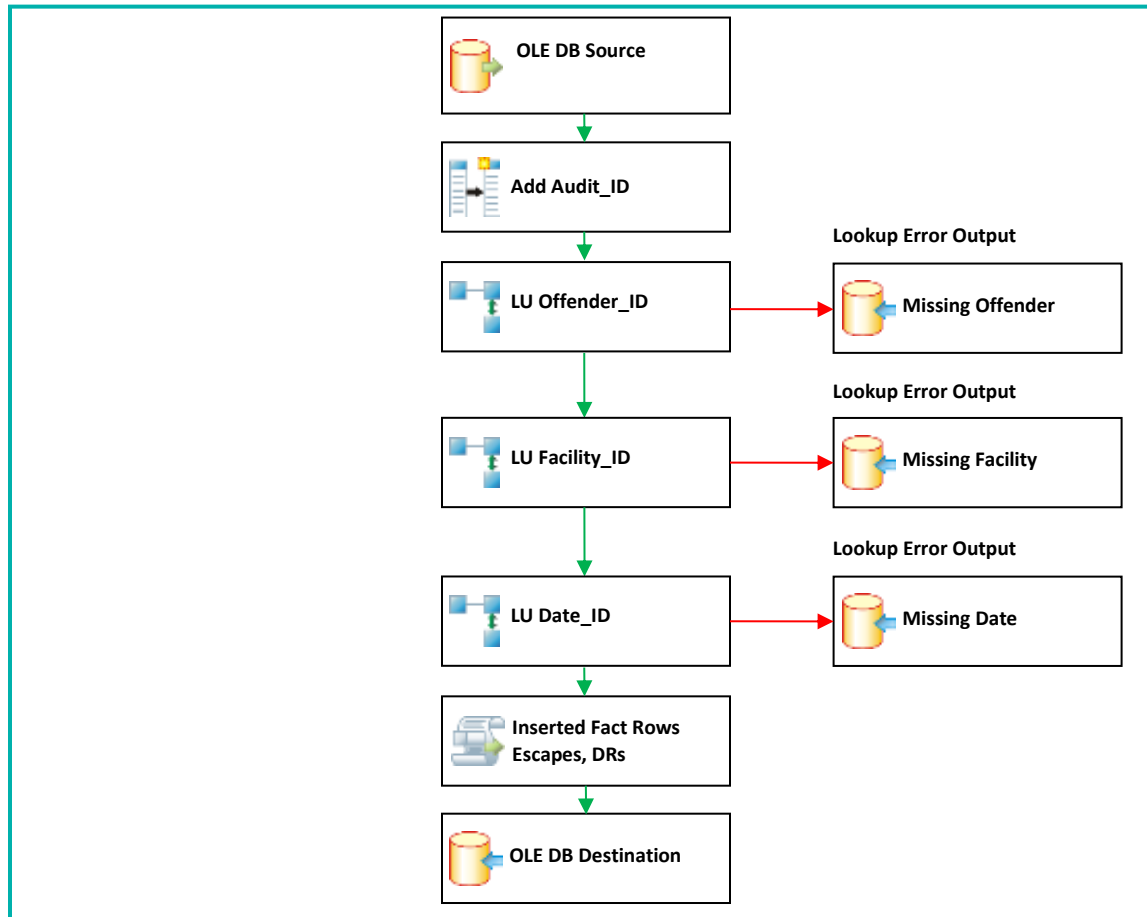


Figure 117: Fact_Event Insert data flow task

Table 47 describes the components of the Fact_Event Insert data flow task.

Table 47: Components of the Fact_Event Insert data flow task		
Symbol	Data flow component	Description
	OLE DB Source	Brings in data from source (REDACTED DCIS) stored procedure usp_FACT_Event
	Add Audit_ID	Derived column transform that adds AuditLogId column to be used in Missing Offenders, Missing Date and MissingFacility tables of the Audit_DB database

Table 47: Components of the Fact_Event Insert data flow task

Symbol	Data flow component	Description
	LU Offender_ID	Look up transform that captures Offender_ID and OffenderKey columns: <pre> SELECT OffenderKey, Offender_ID FROM Dim_Offender WHERE (RowEndDate IS NULL) </pre>
	Missing Offenders	Places look up error output in Missing Offenders Keys table of Audit_DB database
	LU Facility_ID	Look up transform that captures Facility_ID and FacilityCodeKey columns: <pre> SELECT FacilityCodeKey, Facility_ID FROM Dim_Facility WHERE rowenddate is null </pre>
	Missing Facility	Places look up error output in MissingFacility table of Audit_DB database
		Look up transform that captures Date_ID columns <pre> SELECT Date_ID, Date_Name, Year, Year_Name, Quarter, Quarter_Name, Month, Month_Name, Day_Of_Year, Day_Of_Year_Name, Day_Of_Quarter, Day_Of_Quarter_Name, Day_Of_Month, Day_Of_Month_Name, Month_Of_Year, Month_Of_Year_Name, Month_Of_Quarter, Month_Of_Quarter_Name, Quarter_Of_Year, Quarter_Of_Year_Name FROM Dim_Date </pre>
	Missing Date	Places look up error output in Missing Date table of Audit_DB database
	Inserted Fact Rows Escapes, Drs	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	OLE DB Destination	Brings together Escape and DR columns to place in FACT_EVENT table

Investigations, Grievances, UOF

As shown in Figure 118, the Investigations, Grievances and UOF data flow task uses the usp_DWFactEvent stored procedure from the IGLOGS database on the REDACTED server to create the FACT_EVENT table in the data warehouse. This task also handles any errors with the data in Offender_ID, Date_ID and Facility_ID by placing them in tables in the Audit_DB database.

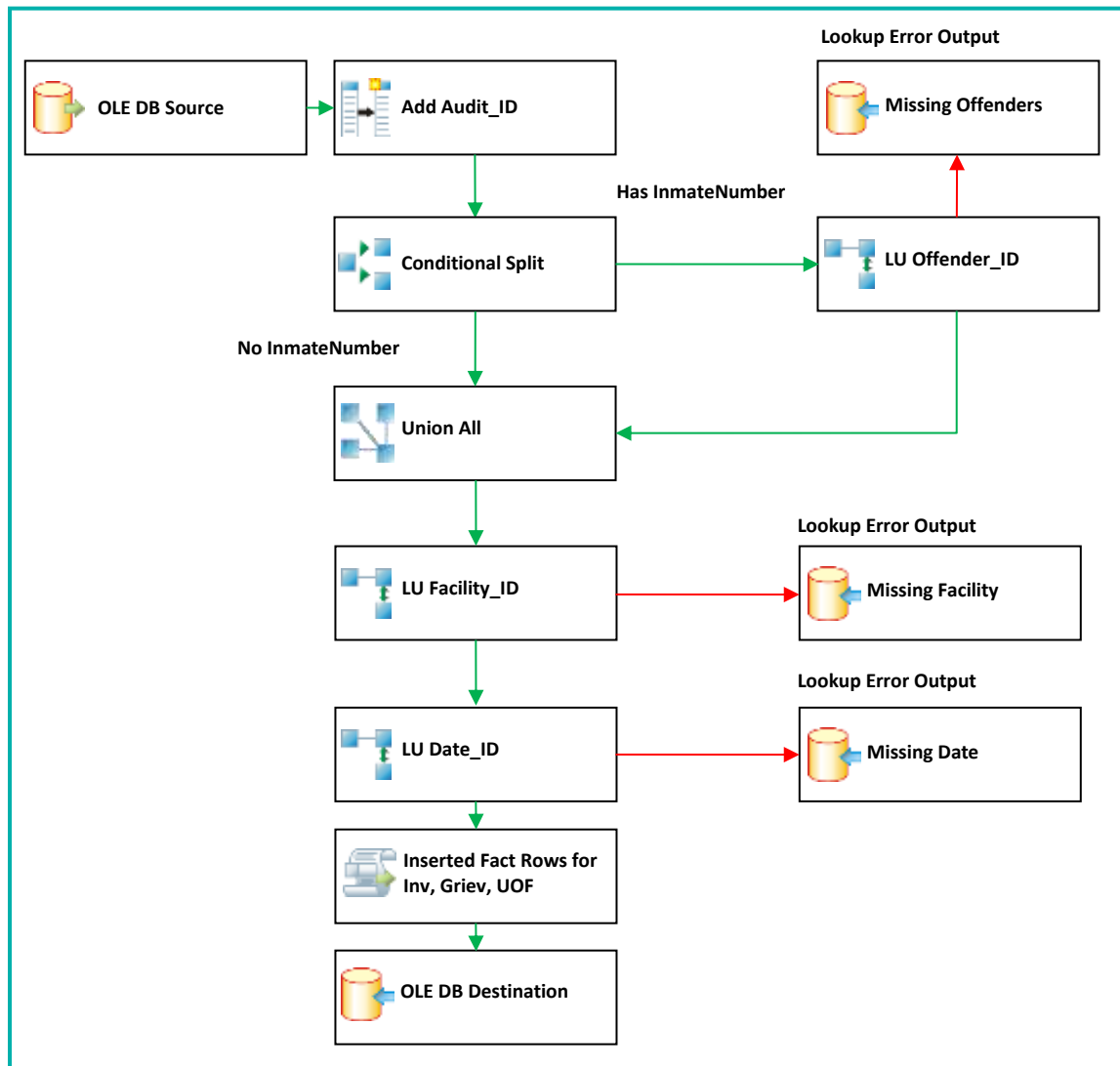


Figure 118: Investigations, Grievances and UOF data flow task

Table 48 describes the components of the Fact_Event Insert data flow task.

Table 48: Components of the Fact_Event Insert data flow task		
Symbol	Data flow component	Description
	OLE DB Source	Brings in data from source (REDACTED IGLOGS) stored procedure usp_DWFactEvent
	Add Audit_ID	Derived column transform that adds AuditLogId column to be used in Missing Offenders, Missing Date and MissingFacility tables of the Audit_DB database
	Conditional Split	Evaluates and redirects the rows where Inmate number is Null (If there is an Inmate number the lookup is performed.)
	LU Offender_ID	Look up transform that captures Offender_ID and OffenderKey columns: <pre>SELECT Offender_ID, BKDCNumber FROM Dim_Offender</pre>
	Missing Offenders	Places lookup error output in Missing Offenders Keys table of Audit_DB database
	Union All	Union all merges the two datasets that result after the conditional split
	LU Facility_ID	Lookup transform that captures Facility_ID and BKOBISFacility columns from Dim_Facility: <pre>SELECT BKOBISFacility, Facility_ID FROM Dim_Facility WHERE RowEndDate IS NULL</pre>
	Missing Facility	Places lookup error output in MissingFacilityIds table of Audit_DB database
		Lookup transform that captures the Date_ID from Dim_Date <pre>SELECT Date_ID FROM Dim_Date</pre>
	Missing Date	Places lookup error output in MissingDateKeys table of Audit_DB database
	Inserted Fact Rows for Inv, Griev, UOF	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	OLE DB Destination	Brings together investigations, grievances and use of force (UOF) columns to be placed in FACT_EVENT table

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 49 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 49: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 SourceIGLOGS		REDACTED	IGLOGS	\\REDACTED\config\SQL\SSIS\SourceIGLOGS.dtsConfig
 SourceDCIS		REDACTED	DCIS	\\REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
 Audit_DB		REDACTED	Audit_DB	\\REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
 DestinationDCDW		REDACTED	DCDW	\\REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

Fact_FacilityActiveInmates

As shown in Figure 119, uses the usp_FACT_FacilityActiveInmates stored procedure the Fact_FacilityActiveInmates to update the Fact_FacilityActiveInmates table and load the DCDW data warehouse on the REDACTED server. In addition, this stored procedure aggregates all the inmate information in the DCDW database to calculate the total number of inmates and gangs in each facility. Finally, it adds auditing data to the ExecutionLog table of the Audit_DB database.

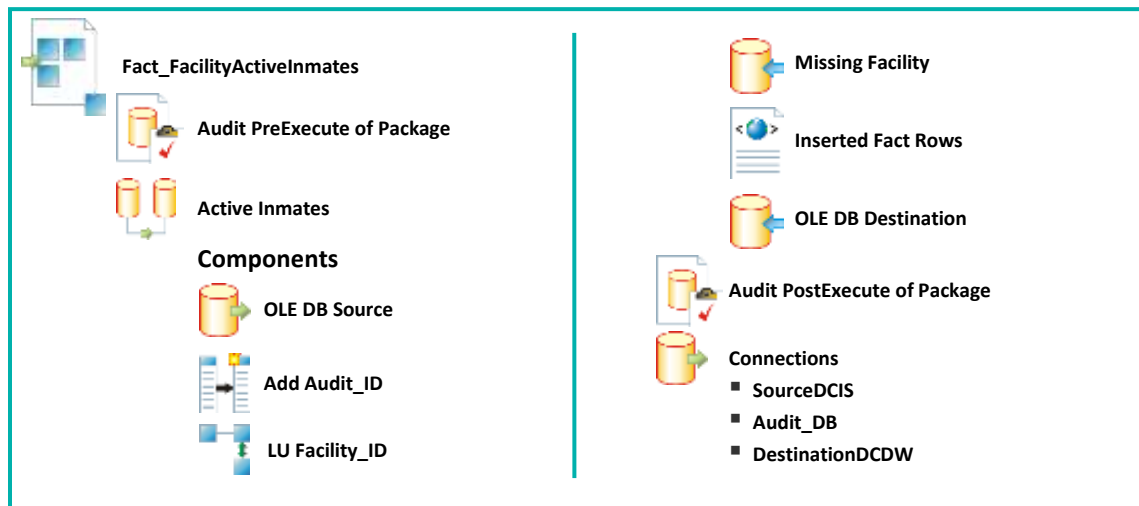


Figure 119: Contents of the Fact_FacilityActiveInmates package

Audit PreExecute of Package

As shown in Figure 120, the Audit PreExecute of Package execute SQL task uses the stored procedure up_Event_Package_OnBegin to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to audit when the package is run, by what machine and by what user. It contains seven input variables and one output variable and runs the following SQL statement:

```
audit.up_Event_Package_OnBegin ?, 'Process Inmate Dimension', ?, ?, ?, ?, ?, ? output
```

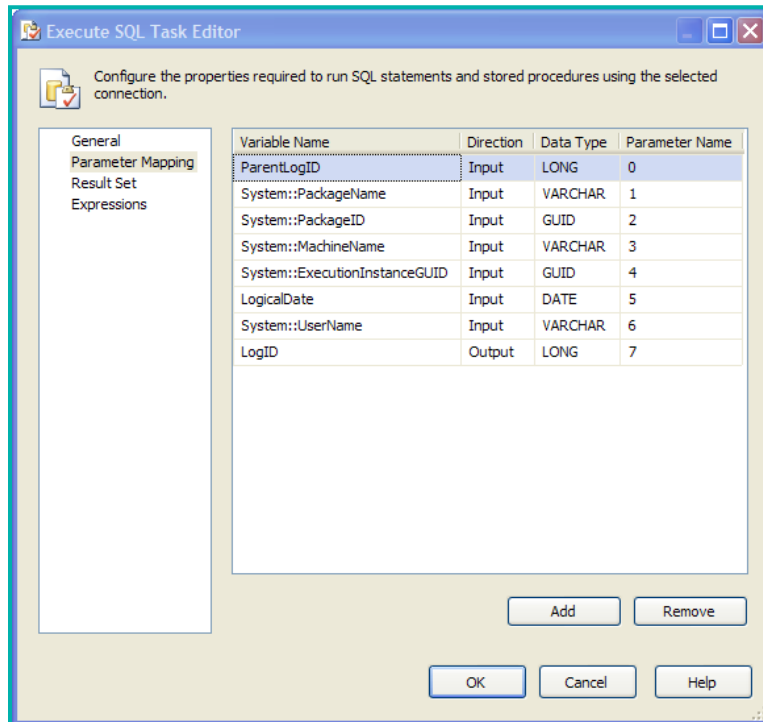


Figure 120: Execute SQL Task Editor dialog box for the Audit PreExecute of Package execute SQL task

Active Inmates

As shown in Figure 121, the Active Inmates flow task uses V_FACT_FacilityActiveInmates view (from the DCIS database on the REDACTED server) to create the FACT_FacilityActiveInmates table in the data warehouse. This task also handles any errors with the data in Facility_ID by placing them in the Audit_DB database tables.

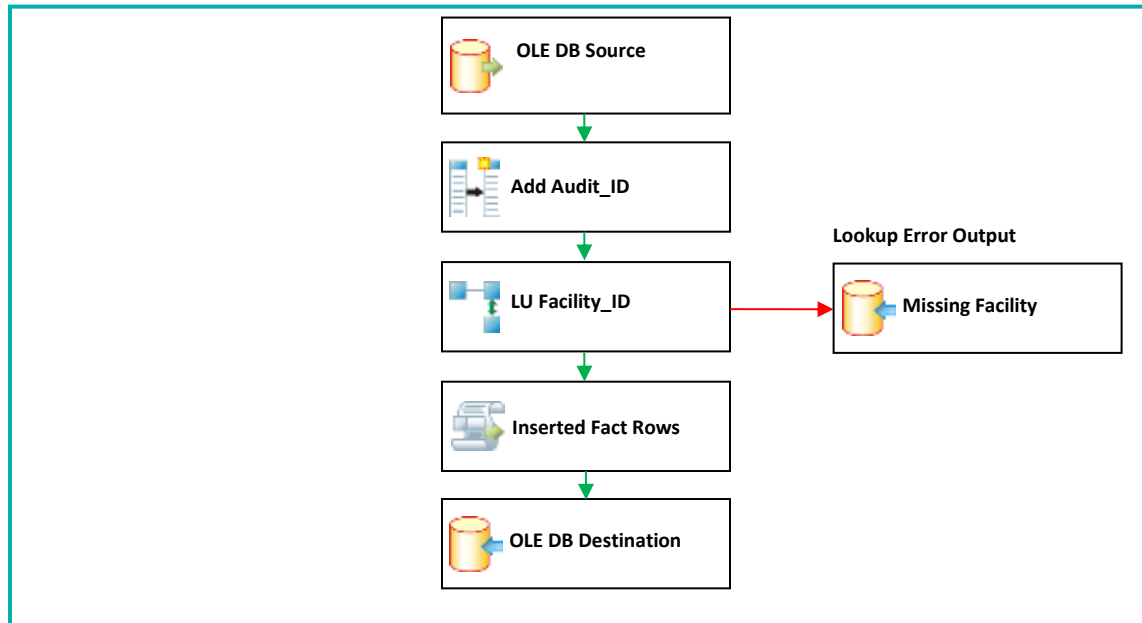


Figure 121: Active Inmates data flow task

Table 50 describes the components of the Active Inmates data flow task.

Table 50: Components of the Fact_FacilityActiveInmates Insert data flow task		
Symbol	Data flow component	Description
	OLE DB Source	Captures the view V_FACT_FacilityActiveInmates from source database DCIS
	Add Audit_ID	Derived column transform that adds AuditLogId column to be used in Missing Offenders and MissingFacility tables of the Audit_DB database
	LU Facility_ID	Lookup transform that captures Facility_ID and FacilityCodeKey columns: <pre> SELECT Facility_ID, BKOBISFacility FROM DIM_Facility WHERE rowenddate is null </pre>
	Missing Facility	Places look up error output in MissingFacility table of Audit_DB database
	Inserted Fact Rows	Custom script transform used to count column rows and places counts in Audit_DB database for auditing purposes
	OLE DB Destination	Brings together needed columns to place in FACT_FacilityActiveInmates table

Audit PostExecute of Package

The Audit PostExecute of Package execute SQL task uses the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 51 lists the connections target servers, target databases and configuration files for the Audit PostExecute of task.

Table 51: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
	SourceDCIS	REDACTED	DCIS	\\REDACTED\config\SQL\SSIS\SourceDCIS.dtsConfig
	Audit_DB	REDACTED	Audit_DB	\\REDACTED\config\SQL\SSIS\AuditDestination.dtsConfig
	DestinationDCDW	REDACTED	DCDW	\\REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

DataMiningFacility

As shown in Figure 122, the DataMiningFacility package updates the Dim_FacilityPredictions table.

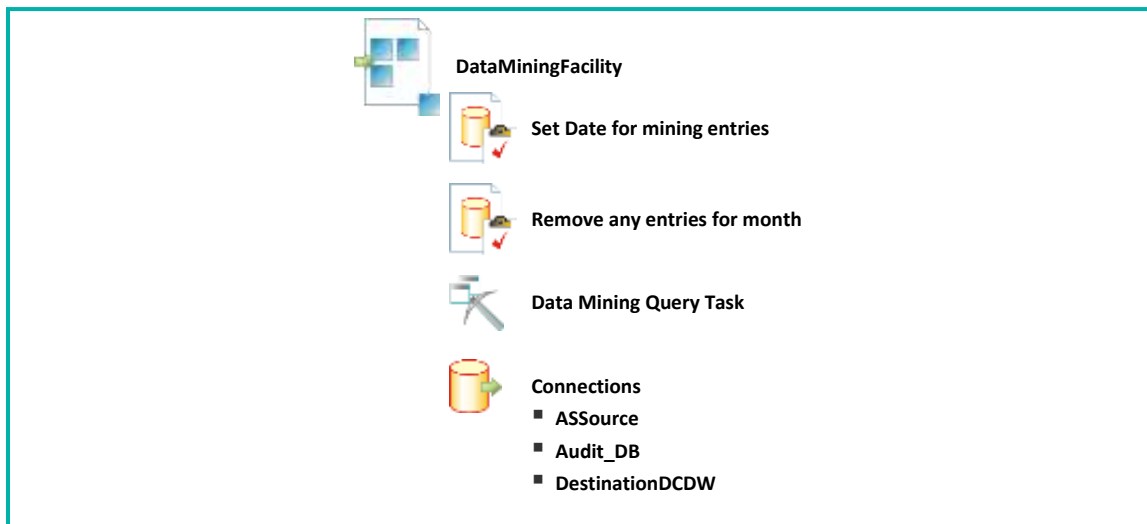


Figure 122: Contents of the DataMiningFacility package

DataMiningFacility

As shown in Figure 123, the DataMiningFacility SSIS package adds records to the Dim_FacilityPredictions table in the DCDW database.

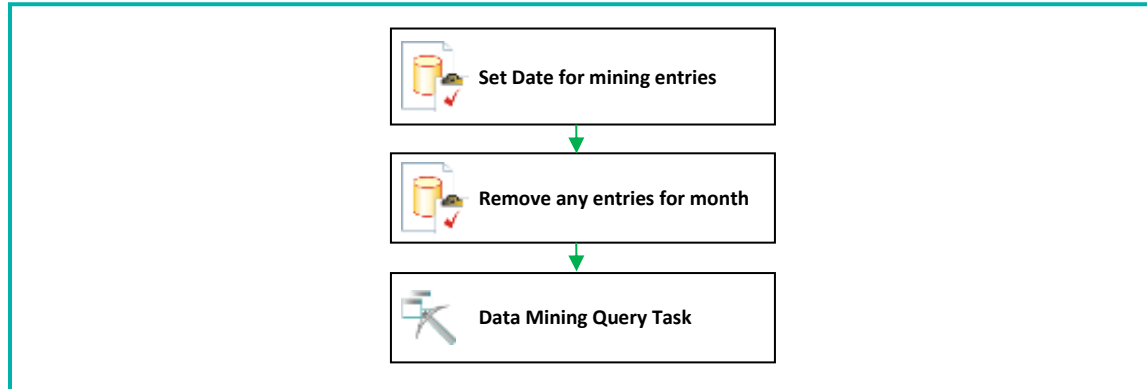


Figure 123: Control flow of the DataMiningFacility SSIS package

Table 52 describes the components of the Active Inmates data flow task.

Table 52: Components of the Fact_FacilityActiveInmates Insert data flow task		
Symbol	Data flow component	Description
	Set Date for mining entries	Selects the first day of the month for the date the package runs: <pre>SELECT convert(datetime,convert(char(2),month (GetDate())) + '/01/' + convert(char(4), year(GetDate())))</pre>
	Remove any entries for month	Since the package will run repeatedly for the current month, current month records must be removed before new records are inserted: <pre>DELETE FROM Dim_FacilityPredictions WHERE datePerformed = ?</pre>
	Data Mining Query Task	Selects facility prediction data from the dmFacility data mining model to populate the Dim_FacilityPredictions table, which is used by the COTAS front end (figures 124 through 128)

Data mining query task

As shown in figures 124 through 128, the data mining query task selects facility prediction data from the dmFacility data mining model to populate the Dim_FacilityPredictions table, which is used by the COTAS front end.

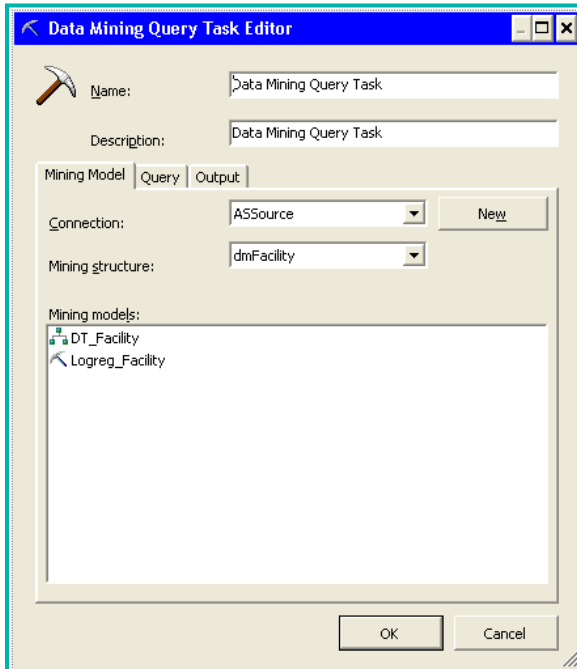


Figure 124: Data Mining Query Task Editor: Mining Model tab

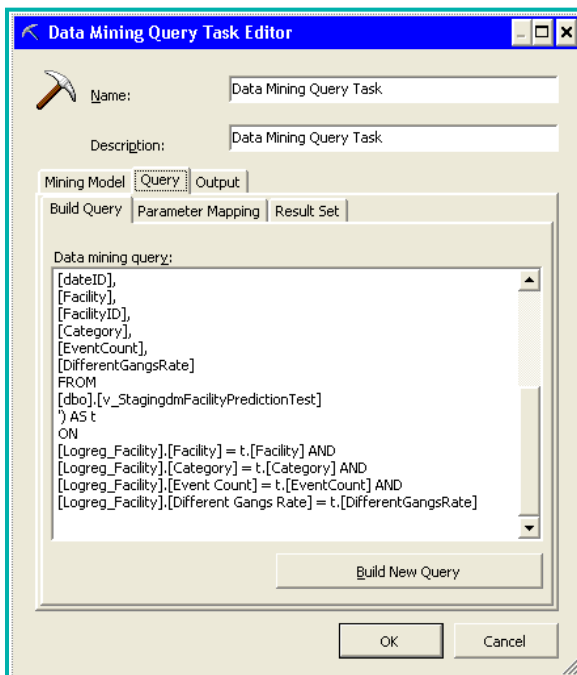


Figure 125: Data Mining Query Task Editor: Query tab

Query

```
SELECT
t.[Dateid] as [datePerformed],
t.[FacilityID] as [Facility_ID],
[Logreg_Facility].[Event Count] as [DifferentGangsRate],
PredictProbability([Logreg_Facility].[Event Count]) as [GangsRatePrediction]
From
[Logreg_Facility]
PREDICTION JOIN
OPENQUERY([DCDW],
'SELECT
[dateID],
[Facility],
[FacilityID],
[Category],
[EventCount],
[DifferentGangsRate]
FROM
[dbo].[v_StagingdmFacilityPredictionTest]
') AS t
ON
[Logreg_Facility].[Facility] = t.[Facility] AND
[Logreg_Facility].[Category] = t.[Category] AND
[Logreg_Facility].[Event Count] = t.[EventCount] AND
[Logreg_Facility].[Different Gangs Rate] = t.[DifferentGangsRate]
```

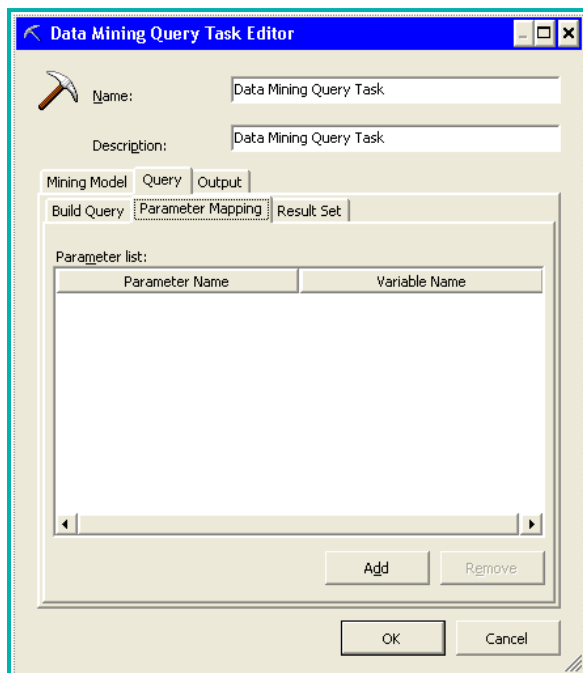
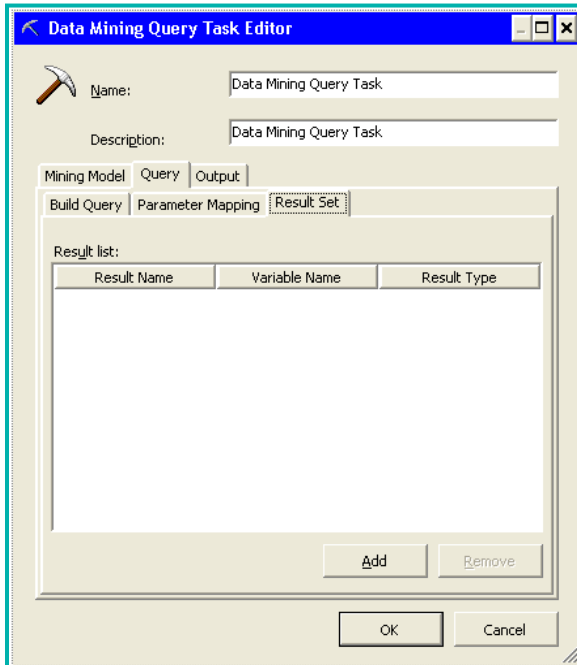


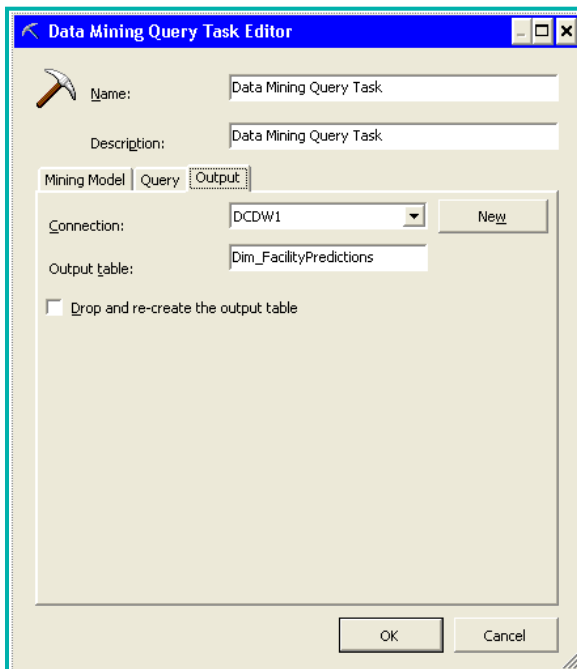
Figure 126: Data Mining Query Task Editor: Parameter Mapping tab



The screenshot shows the 'Data Mining Query Task Editor' window with the 'Result Set' tab selected. The 'Name' and 'Description' fields both contain 'Data Mining Query Task'. Below the tabs, there are three sub-tabs: 'Build Query', 'Parameter Mapping', and 'Result Set'. The 'Result Set' sub-tab is active, displaying a 'Result list' table with three columns: 'Result Name', 'Variable Name', and 'Result Type'. The table is currently empty. At the bottom of the 'Result list' section are 'Add' and 'Remove' buttons. At the bottom of the entire window are 'OK' and 'Cancel' buttons.

Result Name	Variable Name	Result Type
-------------	---------------	-------------

Figure 127: Data Mining Query Task Editor: Result Set tab



The screenshot shows the 'Data Mining Query Task Editor' window with the 'Output' tab selected. The 'Name' and 'Description' fields both contain 'Data Mining Query Task'. Below the tabs, there are three sub-tabs: 'Mining Model', 'Query', and 'Output'. The 'Output' sub-tab is active, displaying fields for 'Connection' (set to 'DCDW1') and 'Output table' (set to 'Dim_FacilityPredictions'). There is a 'New' button next to the 'Connection' dropdown. A checkbox labeled 'Drop and re-create the output table' is present and unchecked. At the bottom of the window are 'OK' and 'Cancel' buttons.

Connection:	DCDW1	New
-------------	-------	-----

Figure 128: Data Mining Query Task Editor: Output tab

Audit PostExecute of Package

The Audit PostExecute task is a SQL task that executes the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table record by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 53 lists the connections target servers, target databases and configuration files for the Audit PostExecute of Package task.

Table 53: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
	ASSource	REDACTED	ASSource	\\REDACTED\config\SQL\SSIS\SourceASSource.dtsConfig
	DestinationDCDW	REDACTED	DCDW	\\REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
	DCDW1	REDACTED	DCDW1	\\REDACTED\config\SQL\SSIS\DestinationDCDW1.dtsConfig

Figures 129 through 131 show the connections target servers, target databases and configuration files for the Audit PostExecute of Package task.

Figure 129: ASSource

Figure 130: DestinationDCDW

Figure 131: DCDW1

DataMiningInmates

As shown in Figure 132, the DataMiningInmates package updates the Dim_InmatePredictions table.

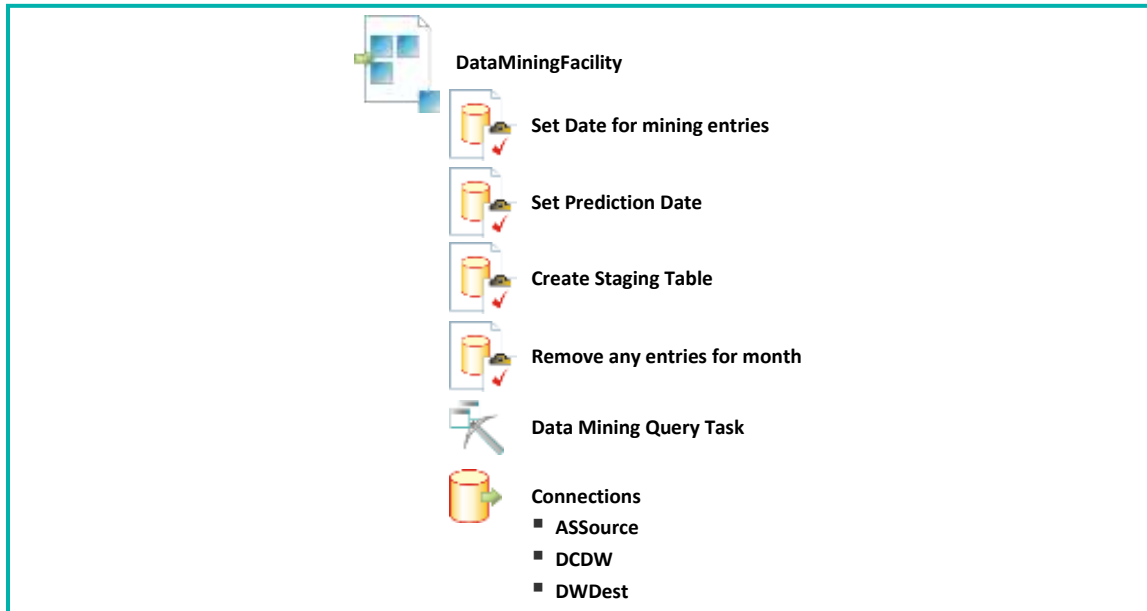


Figure 132: Contents of the DataMiningInmate package

DataMiningInmates

As shown in Figure 133, the DataMiningInmates SSIS package adds records to the Dim_InmatePredictions table in the DCDW database.

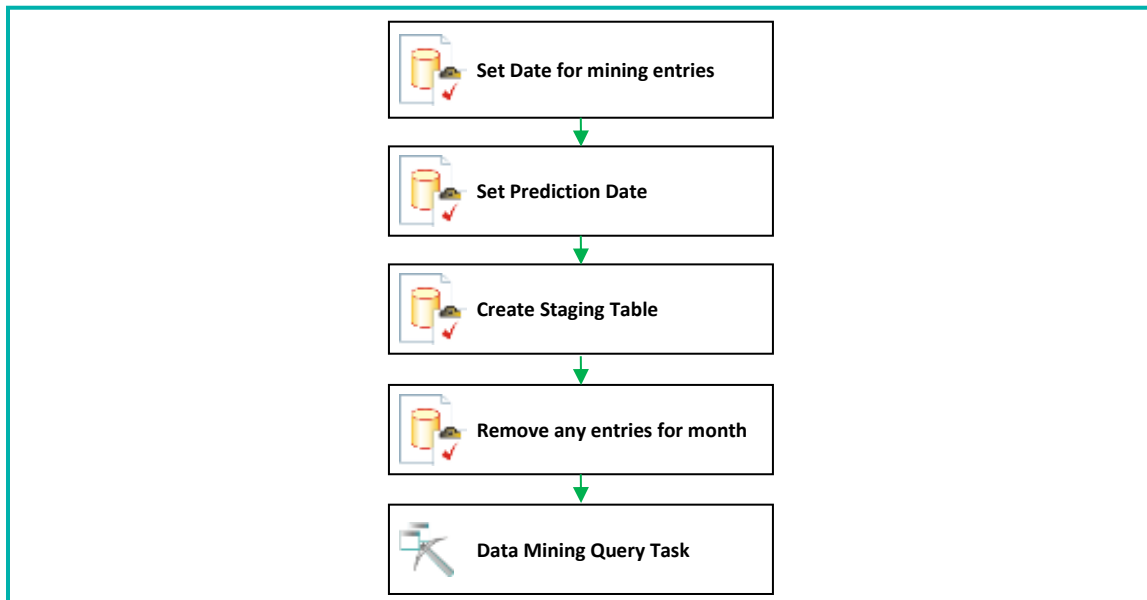


Figure 133: Control flow of the DataMiningInmates SSIS package

Table 54 describes the components of the DataMiningInmates Insert data flow task.

Table 54: Components of the DataMiningInmates Insert data flow task		
Symbol	Data flow component	Description
	Set Date for mining entries	Selects the first day of the month for the date the package runs: <pre>SELECT convert(datetime,convert(char(2),month (GetDate())) + '/01/' + convert(char(4), year(GetDate())))</pre>
	Set Prediction Date	Updates the PredictionDate field in the PredictionSettings with the first day of the month value retrieved in the previous step: <pre>UPDATE [DCDW].[dbo].[PredictionSettings] SET [PredictionDate] =?</pre>
	Create Staging Table	Drops and recreates the StagingdmlInmatePrediction table in DCDW Executes usp_dmlInmatePrediction"
	Remove any entries for month	Since the package will run repeatedly for the current month, current month records must be removed before new records are inserted: <pre>DELETE FROM Dim_InmatePredictions WHERE datePerformed = ?</pre>
	Data Mining Query Task	Selects inmate prediction data from the dmlInmate data mining model to populate the Dim_InmatePredictions table, which is used by the COTAS front end (figures 134 through 138)

Data mining query task

As shown in figures 134 through 138, the data mining query task selects inmate prediction data from the Logreg mining model to populate the Dim_InmatePredictions table, which is used by the COTAS front end.

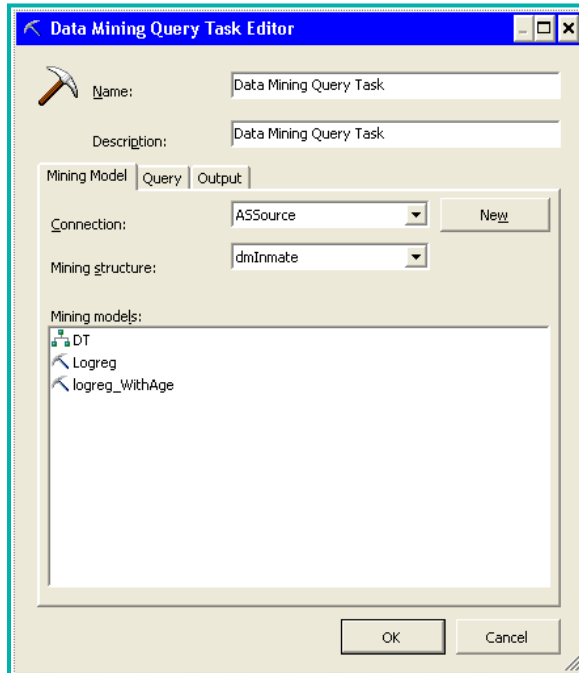


Figure 134: Data Mining Query Task Editor: Mining Model tab

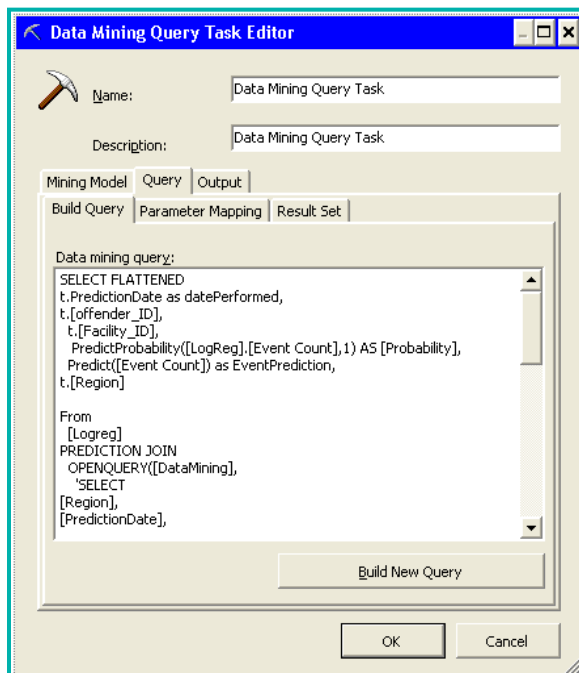


Figure 135: Data Mining Query Task Editor: Query tab

Query

```
SELECT
t.[Dateid] as [datePerformed],
t.[FacilityID] as [Facility_ID],
[Logreg_Facility].[Event Count] as [DifferentGangsRate],
PredictProbability([Logreg_Facility].[Event Count]) as [GangsRatePrediction]
From
[Logreg_Facility]
PREDICTION JOIN
OPENQUERY([DCDW],
'SELECT
[dateID],
[Facility],
[FacilityID],
[Category],
[EventCount],
[DifferentGangsRate]
FROM
[dbo].[v_StagingdmFacilityPredictionTest]
') AS t
ON
[Logreg_Facility].[Facility] = t.[Facility] AND
[Logreg_Facility].[Category] = t.[Category] AND
[Logreg_Facility].[Event Count] = t.[EventCount] AND
[Logreg_Facility].[Different Gangs Rate] = t.[DifferentGangsRate]
```

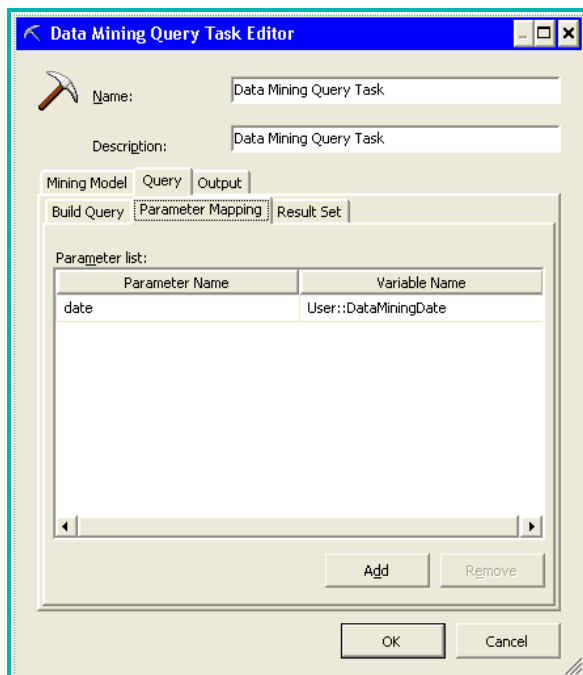


Figure 136: Data Mining Query Task Editor: Parameter Mapping tab

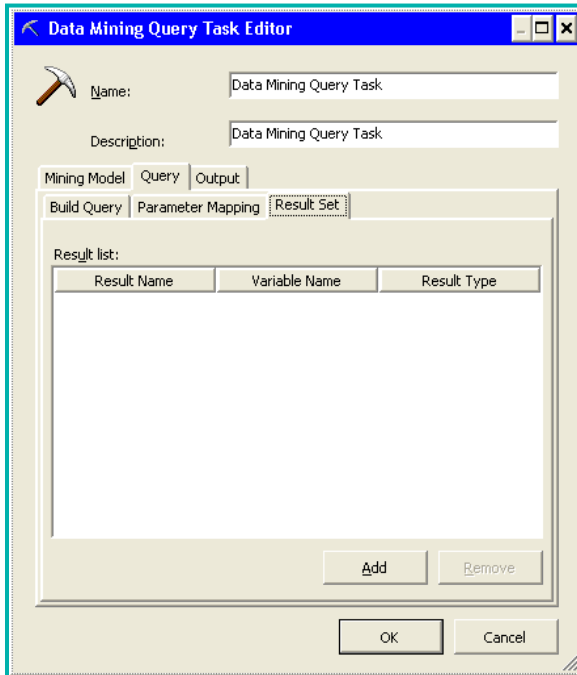


Figure 137: Data Mining Query Task Editor: Result Set tab

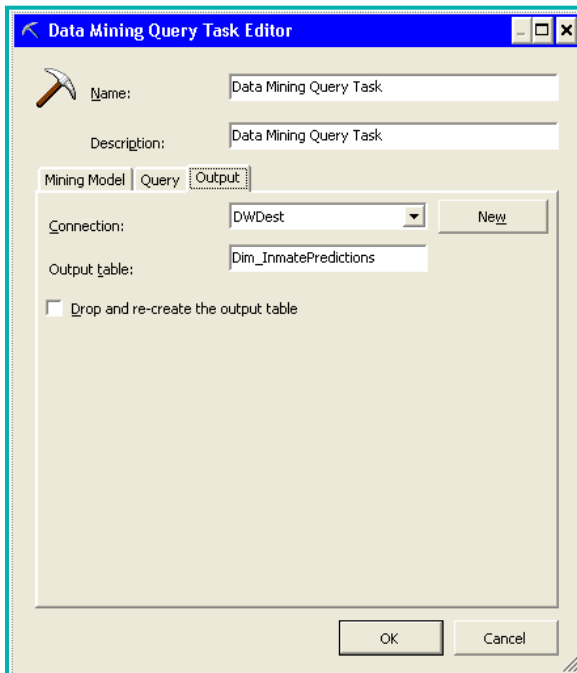


Figure 138: Data Mining Query Task Editor: Output tab

Audit PostExecute of Package

The Audit PostExecute task executes the stored procedure audit.up_Event_Package_OnEnd to add auditing data to the table ExecutionLog in the Audit_DB. The purpose of this task is to complete the audit table record by adding the time the package completed in the column EndTime. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 55 lists the connections target servers, target databases and configuration files for the Audit PostExecute of Package task.

Table 55: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
	ASSource	REDACTED	ASSource	\\REDACTED\config\SQL\SSIS\SourceASSource.dtsConfig
	DWDest	REDACTED	DWDest	\\REDACTED\config\SQL\SSIS\DestinationDCDest.dtsConfig
	DestinationDCDW	REDACTED	DCDW	\\REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig

Figures 139 through 141 show the connections target servers, target databases and configuration files for the Audit PostExecute of Package task.

Figure 139: ASSource

Figure 140: DWDest

Figure 141: DCDW

ProcessCube

As shown in figures 142 and 143, the ProcessCube SSIS package processes the dimensions and cube in the DOC_Events OLAP database.

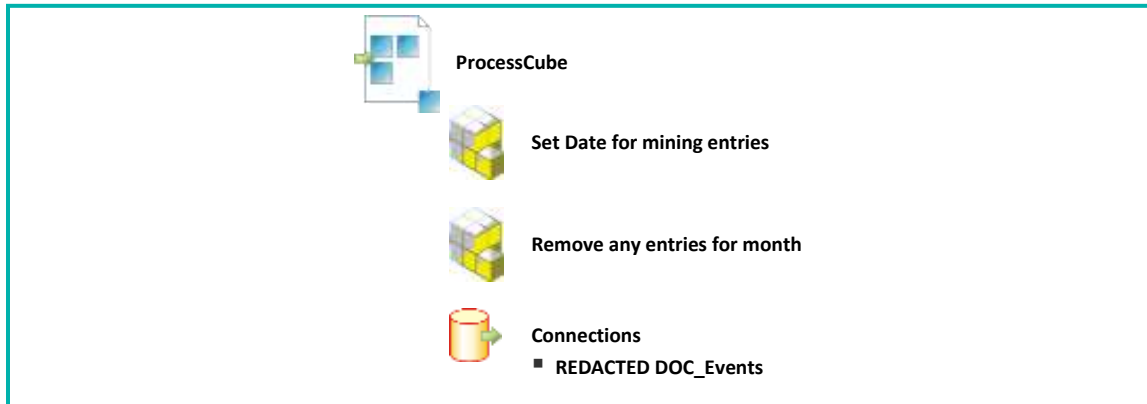


Figure 142: Contents of the ProcessCube package

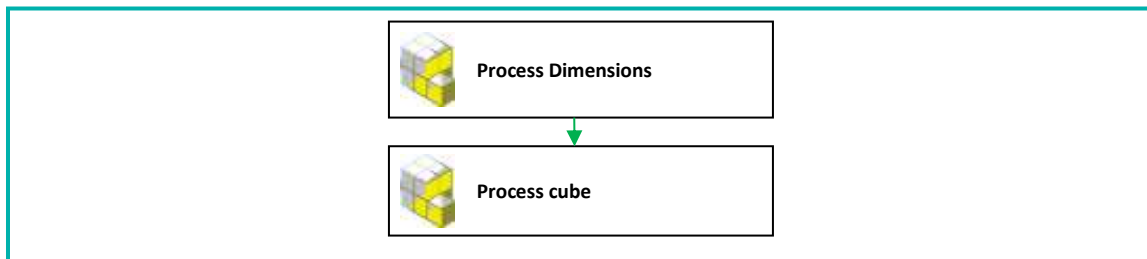


Figure 143: Control flow of the ProcessCube SSIS package

Process dimensions

As shown in figures 144 through 148, the Process Dimensions task processes all the cube's dimensions.

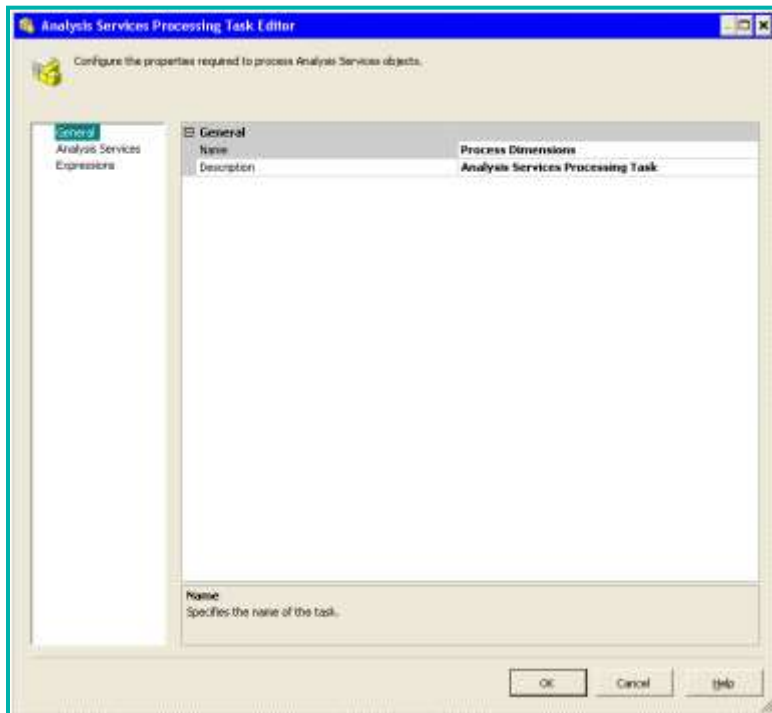


Figure 144: Analysis Services Processing Task Editor: General page for process dimensions

Figure 145: Analysis Services Processing Task Editor: Analysis Services page

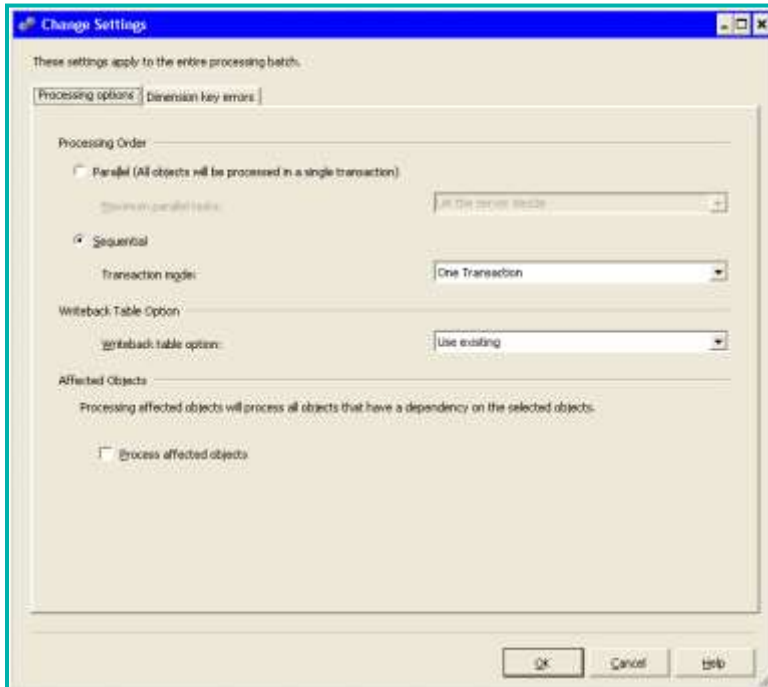


Figure 146: Analysis Services Processing Task Editor: Change Settings page, Processing options tab

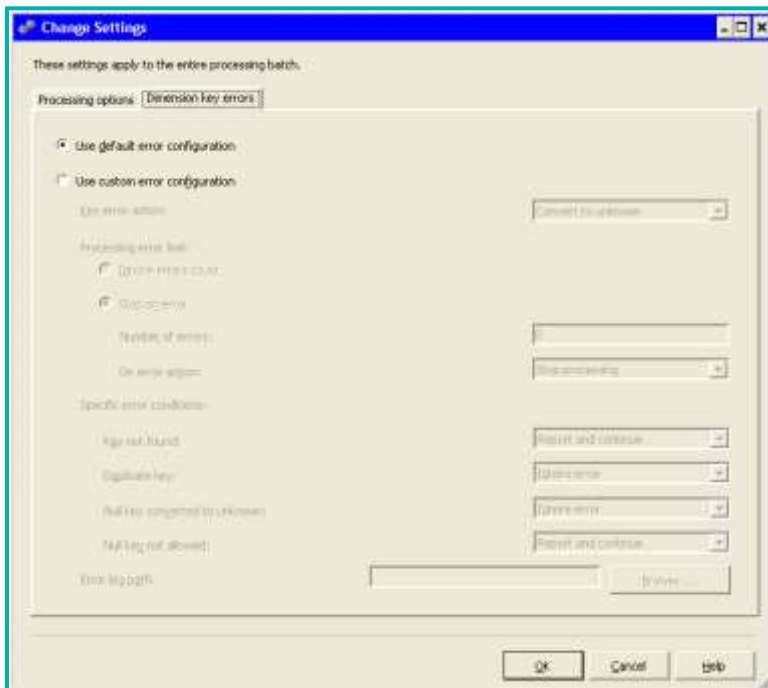


Figure 147: Analysis Services Processing Task Editor: Change Settings page, Dimension key errors tab

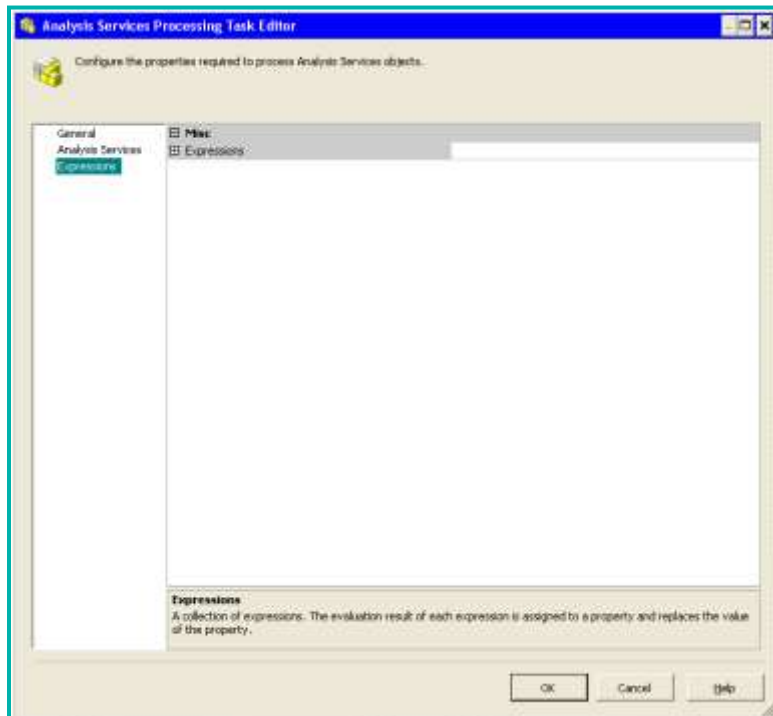


Figure 148: Analysis Services Processing Task Editor: Expressions page

Process cube

As shown in figures 149 through 153, the Process Cube task processes the cube.

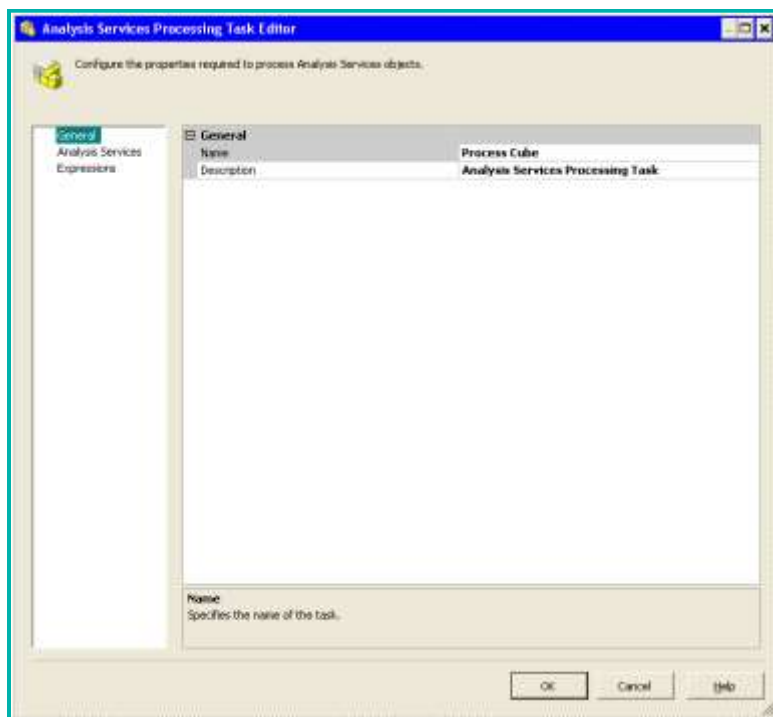


Figure 149: Analysis Services Processing Task Editor: General page for the Process Cube

Figure 150: Analysis Services Processing Task Editor: Analysis Services page

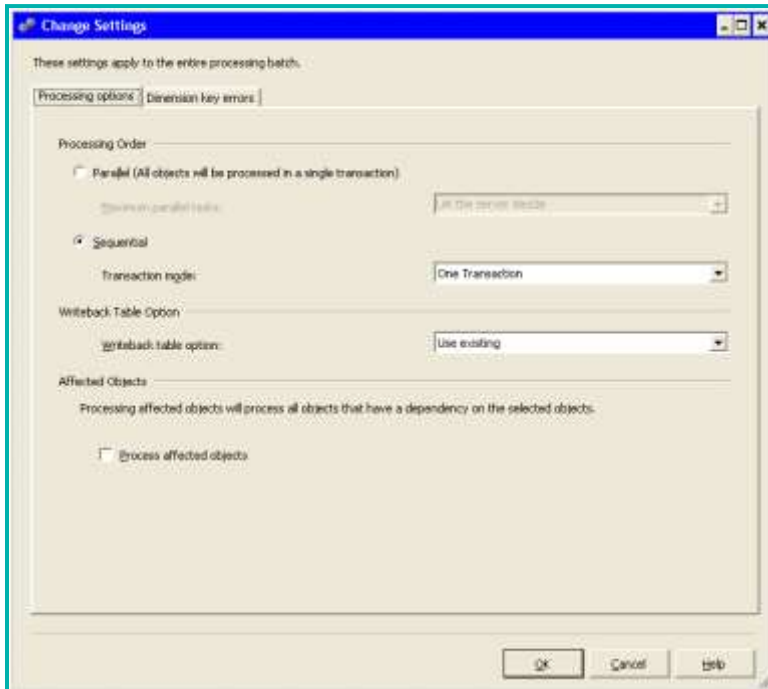


Figure 151: Analysis Services Processing Task Editor: Change Settings page, Processing options tab

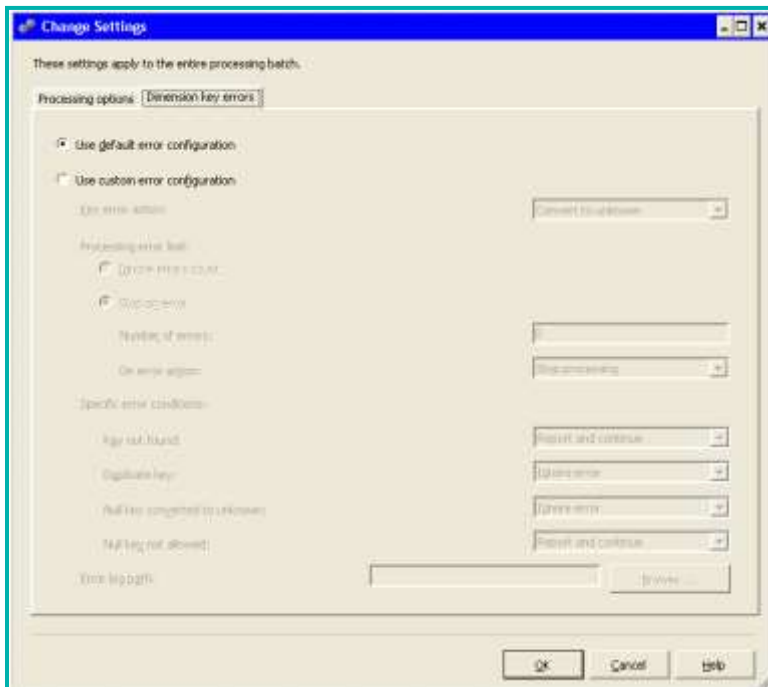


Figure 152: Analysis Services Processing Task Editor: Change Settings page, Dimension key errors tab

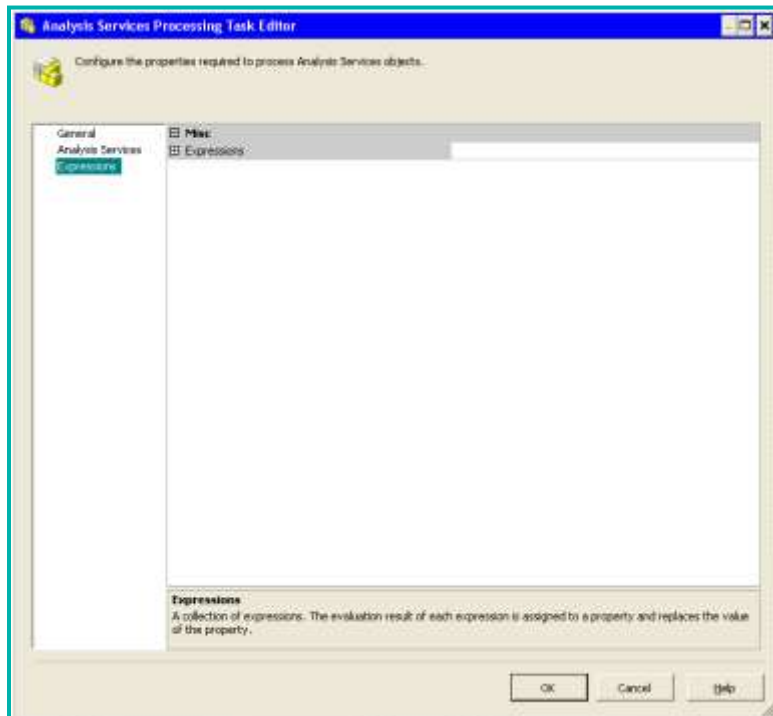


Figure 153: Analysis Services Processing Task Editor: Expressions page

Data mining query task

As shown in figures 154 through 158, the data mining query task selects facility prediction data from the dmFacility data mining model to populate the Dim_FacilityPredictions table, which is used by the COTAS front end.

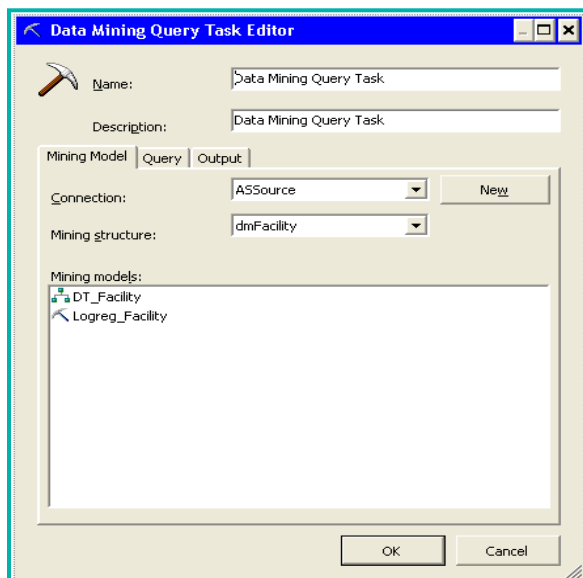


Figure 154: Data Mining Query Task Editor: Mining Model tab

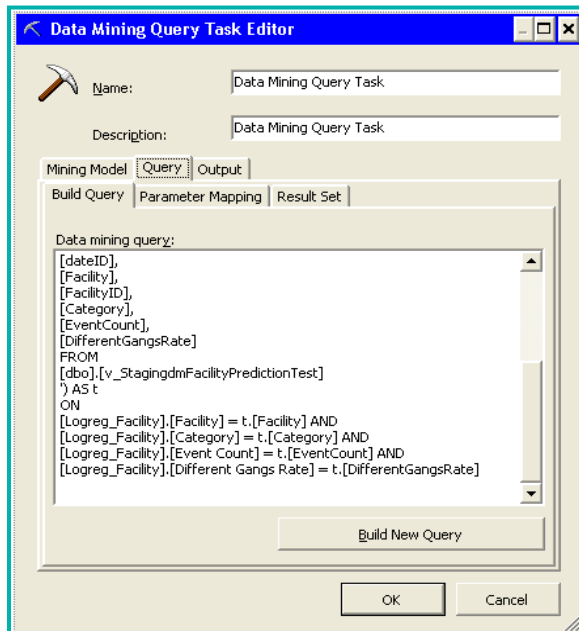


Figure 155: Data Mining Query Task Editor: Build Query tab

Query

```
SELECT
t.[Dateid] as [datePerformed],
t.[FacilityID] as [Facility_ID],
[Logreg_Facility].[Event Count] as [DifferentGangsRate],
PredictProbability([Logreg_Facility].[Event Count]) as [GangsRatePrediction]
From
[Logreg_Facility]
PREDICTION JOIN
OPENQUERY([DCDW],
'SELECT
[dateID],
[Facility],
[FacilityID],
[Category],
[EventCount],
[DifferentGangsRate]
FROM
[dbo].[v_StagingdmFacilityPredictionTest]
') AS t
ON
[Logreg_Facility].[Facility] = t.[Facility] AND
[Logreg_Facility].[Category] = t.[Category] AND
[Logreg_Facility].[Event Count] = t.[EventCount] AND
[Logreg_Facility].[Different Gangs Rate] = t.[DifferentGangsRate]
```

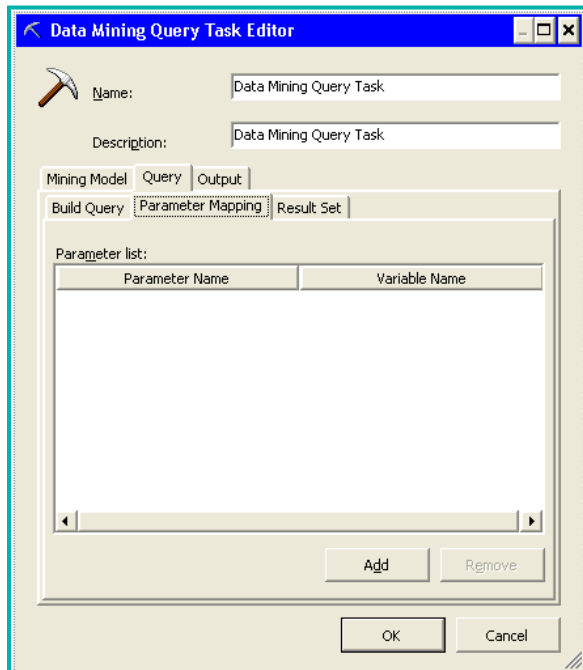


Figure 156: Data Mining Query Task Editor: Parameter Mapping tab

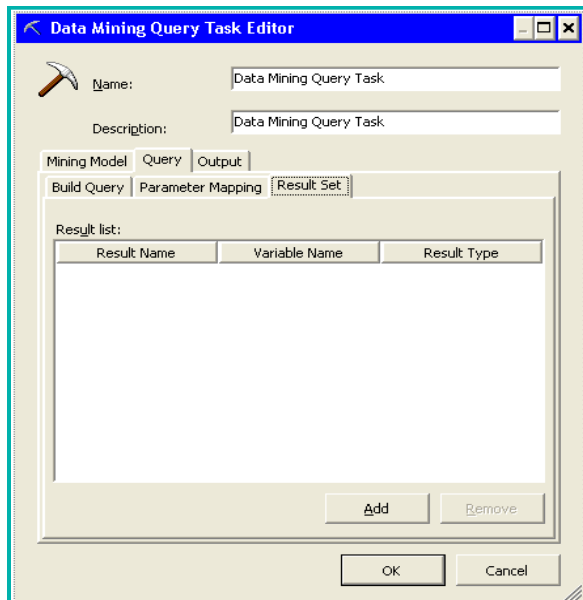


Figure 157: Data Mining Query Task Editor: Parameter Result Set tab

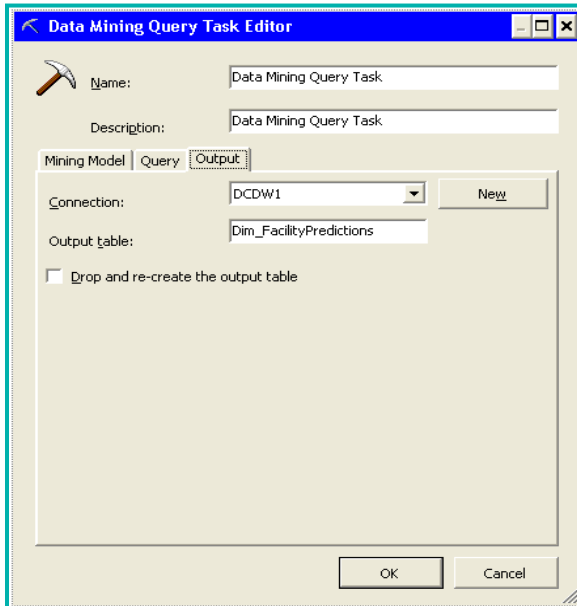


Figure 158: Data Mining Query Task Editor: Parameter Output tab

Audit PostExecute of Package

The Audit PostExecute task executes the stored procedure `audit.up_Event_Package_OnEnd` to add auditing data to the table `ExecutionLog` in the `Audit_DB`. The purpose of this task is to complete the audit table record by adding the time the package completed in the column `EndTime`. In addition, this task runs the following SQL statement:

```
audit.up_Event_Package_OnEnd @logID = ?
```

Table 56 lists the connections target servers, target databases and configuration files for the Audit PostExecute of Package task.

Table 56: Connections for the Audit PostExecute of Package task				
Connection		Target server	Target database	Configuration file
 ASSource		REDACTED	ASSource	\\REDACTED\config\SQL\SSIS\SourceASSource.dtsConfig
 DestinationDCDW		REDACTED	DWDest	\\REDACTED\config\SQL\SSIS\DestinationDCDW.dtsConfig
 DCDW1		REDACTED	DCDW1	\\REDACTED\config\SQL\SSIS\DCDW1.dtsConfig

Figures 159 through 161 show the connections target servers, target databases and configuration files for the Audit PostExecute of Package task.

Figure 159: ASSource connection



Figure 160: DestinationDCDW connection



Figure 161: DCDW1 connection

Appendix B – COTAS Tutorial

Identifying Issues before they develop

The Correctional Operations Trend Analysis System (COTAS) consolidates historical information with a predictive scoring system and makes it readily available through dashboards and reports. This information is presented in ways that increase the likelihood of spotting problems that might otherwise go unnoticed for sometime in organizations where information is unconsolidated and no predictive scoring system exists. Once you spot a concern, you can conduct deeper analysis to determine if changes in personnel, inmates, physical surroundings or other variables might be indicated to alleviate an existing issue or avoid future issues.

COTAS can only consolidate information that an organization's systems collect for the management of inmates and facilities. Consequently, some items shown may not be available for some organizations. The following sections provide a tour of the various areas of the application from the perspective of some commonly found positions in the operation of correctional systems and institutions.

COTAS presents two types of information: historical and predictive. Consequently, there are two paths one can take from the **RegionGauges** dashboard. In Figure , the gauges at the top half of the screen represent historical information for the last 30 days; the gauges at the bottom half of the screen represent predictive information. This tutorial will take the two separate paths. First the historical path will be explored from the perspective of three types of users within a corrections facility. The predictive path will be explored from the inmate and facility perspectives.

Figure 162: Historical and predictive gauges by region

Conducting Historical Analysis

The starting point in the COTAS system is the **RegionGauges** dashboard (Figure). From this dashboard, you can investigate the following key, regional values in the effective operation of institutions:

- Number of violent events
- Predicted violent events
- Central office grievances
- Disciplinary reports
- Field grievances
- Investigations
- Use of force
- High-profile inmates.

Note: Inmate search is also available from the **RegionGauges** dashboard.

Investigating events of interest

If you notice an event that stands out as worthy of further investigation, you can often click the item to drill deeper into the details. Upon further investigation, you may find other items that are worthy of investigation. Frequently, there are additional details available through further drilling into the data by clicking charts or hyperlinks. Some areas allow you to drill down and identify individual inmate reports and profiles. As with any organization, different people in the organization are interested in different levels of detail. COTAS is secured in a way that allows individuals access to areas and screens that they have been preauthorized to see based on their positions.

Monitoring facilities (chief security officers)

Note: This section discusses the gauges on the historical, top half of the screen. The gauges on the predictive lower half of the screen are discussed later.

A typical scenario for a chief security officer of an entire department of corrections might be to look at the **RegionGauges** dashboard (Figure). This dashboard displays a high-level regional rollup by region for every facility in the system. Like a stop light, the region gauges use the following color scheme:

- Green: All systems go
- Yellow: Caution
- Red: Stop and take notice

No further investigation is necessary if everything is in the acceptable zones (green). If anything is outside of the acceptable zones (yellow or red), you can click the appropriate items and drill deeper into the details of a facility.

To monitor facilities by region:

- 1 In the top row of the **RegionGauges** screen, under **Last Month's Facilities with Violent Events Exceeding Thresholds** (Figure), click the region link you want to monitor (**Region 3** in this case).

*COTAS displays the **RegionGraph** screen for Region 3 (Figure 163).*

Note: The **RegionGraph** screen shows detailed trending of the violent events being tracked including disciplinary reports, investigations and use of force. This trend information can provide important insights into the functioning of the region over the past year, although it will be necessary to know more information about the numbers at specific facilities to identify where action should be taken.

Figure 163: RegionGraph screen for Region 3

- 2 Return to the Region Gauges screen (Figure 16).

Figure 164: Historical and predictive gauges by region

- 3 Click the **Region** gauge instead of the **Region** link (in this case, Region 3).

*COTAS opens the **RegionDashboard** in Figure .*

Note: The **RegionDashboard** screen provides details about specific, Region 3 facilities. The determination of whether a facility's violent event count falls into red, yellow or green depends on thresholds that have been set up for each facility based on its type and category—not just event count. This feature allows for the weighting of facilities so they can be measured against realistic expectations. Having all of this logic built into the system ensures that one's attention is drawn to facilities that need it most. To the right of the individual facility information is a map showing its location, a summary chart for events by event type and a summary chart for events by facility category.

Figure 165:RegionDashboard for Region 3

- 4 Click the name of the facility to review additional details for a specific facility (in this case, **Brevard C.I.**).

*COTAS opens the **FacilityMonitor** screen for Brevard C.I. (Figure166).*

Note: The **FacilityMonitor** screen is a facility-specific dashboard that shows violent and non-violent events over the last 30 days and breaks events down by disciplinary reports, escapes, investigations and use of force.

Figure166: FacilityMonitor screen

After reviewing the screens in Figures **Error! Reference source not found.** through 166, the Chief Security Officer usually contacts the Warden or other supervisory staff at the facility to discuss these findings. The Warden is usually aware of all situations in the facility and has answers ready when the Chief Security Officer calls. The next section discusses the screens wardens or other supervisory staff might use to stay apprised of events occurring at their facilities.

Reviewing facility events (wardens)

A typical scenario for wardens might be to start at the **FacilityMonitor** screen (Figure) for their facility. This screen is a dashboard for a facility that shows the violent and non-violent events for the past 30 days, and further breaks the events down by disciplinary reports, escapes, investigations and use of force. If everything is in the acceptable zone, no further investigation may be necessary. The **FacilityMonitor** screen offers wardens three options for drilling deeper to see trend graphs or details about individual incidents: an event graph screen, a disciplinary report screen and an event summary by dorm screen.

Figure 167: FacilityMonitor screen

To review the facility monitor report:

- 1 In the upper-left of corner of the **Violent** or **Non-Violent** section (Figure), click the facility name (in this case, click **Brevard C. I.** in the **Violent** section).

*COTAS displays the **EventGraph** screen (Figure), which shows the entire facility's trends for all types of violent or non-violent events over the last year.*

Figure 168: Violent event trends on the EventGraph screen for Brevard C.I.

- 2 Above gauges such as **Disciplinary Reports** and **Escapes**, click the event type links to drill deeper into the details of a specific kind of violent or non-violent event.

*Clicking the event type link over **Disciplinary Reports** in the **Violent** section generates the screen in Figure , which shows the trend for the disciplinary report event type on violent and non-violent events over the last year.*

Figure 169: Disciplinary report event type trend on the Event Graph screen for Brevard C.I.

- 3 In the **Violent** or **Non-Violent** section, click the gauge to drill down to the event count to see a list of the events that make up the count shown on the gauge.

*Clicking the **Disciplinary Reports** gauge in the **Violent** section generates the **EventSummarybyDorm** screen in Figure .*

Figure 170: EventSummerybyDorm screen, Events tab

Note: The **EventSummarybyDorm** screen includes the following tabs:

- Events
- BedMission
- Dorm
- Primary Work Assignment
- Description

The name of the tab indicates how the event information is grouped. When the **EventSummarybyDorm** screen appears, it defaults to the **Events** tab, which shows a list of violent events grouped by event number. By clicking the tabs on the **EventSummarybyDorm** screen, you can see if events are more concentrated in a particular bed mission, dorm, primary work assignment or event description (Figures **Error! Reference source not found.** through **Error! Reference source not found.**). Sorting by any column is generally available on list screens.

- 4 Click through the **Events**, **BedMission**, **Dorm**, **Primary Work Assignment** and **Description** tabs (Figures 170 through 174).

Figure171: EventSummerybyDorm screen, BedMission tab

Figure 172:EventSummerybyDorm screen, Dorm tab

Figure 173: EventSummerybyDorm screen, Primary Work Assignment tab

Figure174: EventSummerybyDorm screen, Description tab

Drilling into event details (operational staff)

At this point, the Warden may feel that the physical surroundings and the staff need no rearrangement and requests that one of the operational staff bring detailed information about the incidents and inmates involved to discussed moving them. A typical scenario may involve a supervisor or assistant supervisor. The Warden might review the events of the previous day on the **EventSummarybyDorm** screen.

Follow these steps to drill into event details:

- 1 On the **EventSummarybyDorm** screen, click the **Event** expansion link you want to investigate.

*The **EventSummarybyDorm** screen expands to show the high-level details of the event (Figure).*

Figure175: Events expanded on the EventSummarybyDorm screen

- 2 Click the **Log Number** link.

*COTAS displays **Disciplinary Report/Investigation** screen (Figure).*

Figure 176:Disciplinary Report/Investigation screen

Note: From the **Disciplinary Report/Investigation** screen, you can review the complete details of the event on the disciplinary report. The **Disciplinary Report/Investigation** screen is linked to the existing inmate management system through the inmate's picture.

- 3 Click the inmate's picture.

COTAS retrieves the standard inmate information from the existing inmate management system (Figure).

Figure 177: Standard inmate information from the Corrections Offender Information Network

Predicting and mitigating risk

This section discusses the predictive graphs on the bottom half of the **RegionGauges** screen. Like the historical path, the predictive path begins at the region gauges (Figure).

Note: Figure is the same as Figure but is duplicated here for convenience. This section does not show the starting points for users at different levels of a correctional organization the points in COTAS, because they were shown in the historical section. In addition, the predictive path has fewer screens than the historical path. Consequently, this section takes a more direct approach to explaining the predictive screens.

Figure 178:RegionGauges screen, the starting place for the predictive path

The predictive graphics are often referred to as life savers, because they bear close resemblance to the Life Savers candy. The green-yellow-red color scheme continues with the predictive graphics, because they display the relative amount of risk for violent events going from green to red. In the predictive section, clicking either a region name link above the graphic such as **Region 3** has the same effect as clicking the graphic itself. They both generate the **RegionPredictorDashboard** screen in Figure .

Assessing risk with the Inmate Predictor

The Inmate Predictor enables you to drill down from regions to facilities and specific inmates to investigate information such as non-violent events, violent events and time served. You can also compare an inmate's information with peers who have same characteristics to calculate a risk score.

Follow these steps to investigate risk with the Inmate Predictor:

- 1 In the bottom row of the **RegionGauges** screen, under **Future Predicted Violent Events Based on Inmates** (Figure), click the region graph or link (**Region 3** in this case).

*COTAS displays the **RegionPredictorDashboard** screen for Region 3 (Figure).*

Note: The **RegionPredictorDashboard** screen opens two new paths of information: the **Inmate Predictor** path and the **Facility Predictor** path.

Figure 179: Facility Predictor image and link on the RegionPredictorDashboard screen

- 2 Under **Region 3 Predictors**, click **Inmate Predictor** graph or link.

*COTAS displays the **RegionPredictorList** screen (Figure).*

Figure 180: RegionPredictorList screen showing levels of risk for violent events

Note: The **RegionPredictorList** screen shows relative amounts of the facility's violence risk with increasing amounts of red. Yellow represents less risk, but may be cause for concern.

Green represents substantially less risk. In the Inmate Predictor path, risk is based on aggregated inmate risk scores at the facility.

- 3 Click the **Facility** link to drill deeper into the details of the potential risk (**Brevard C.I.** in this case).

*COTAS displays the **PredictorSummaryByDorm** screen (Figure).*

Note: The **PredictorSummaryByDorm** screen defaults to the **Bed** tab, which shows information by bed numbers, by extension and individual inmates. The far-right column displays the risk score, which is calculated with advanced predictive algorithms that factor in the inmates number of violent events, months since last violent event, race, age, gender and time served. These characteristics are correlated with the historical violence tendencies of others with the same characteristics to calculate a risk score. A risk score with an orange color indicates an inmate that has transferred into the unit within the past 30 days.

Figure 181: PredictorSummaryByDorm screen

- 4 Mouse over a risk score for one or two seconds.

COTAS displays a screen tip with inmate information such as number of violent and non-violent events, time since last violent event and date transferred into unit (Figure).

Figure 182: Screen top with inmate information

- 5 Click through the **BedMission**, **Dorm** and **Primary Work Assignment** tabs.

Note: As with similarly structured screens in the historical path, the **BedMission**, **Dorm** and **Primary Work Assignment** tabs display the same information with the columns rearranged and grouped by the item indicated by the name of the tab as shown in Figures 183-185.

Figure 183: BedMission tab of the PredictorSummaryByDorm screen

Figure 184: Dorm tab of the PredictorSummaryByDorm screen

Figure 185: Primary Work Assignment tab of the PredictorSummaryByDorm screen

- 6 Click the **BedMission** tab.
- 7 Click the bed mission expansion link to drill into the bed mission the detail (in this case, **Child Nutrition** expansion link).

The expansion link displays the bed mission detail (Figure).

Note: In Figure , the **Child Nutrition** bed mission is expanded to show how you can drill into the details of a typical bed mission.

Figure 186: Child Nutrition bed mission expanded in the BedMission tab

- 8 Click a **Risk** link (in this case, the top score: **87.89** in Figure).
COTAS displays the inmate's information (Figure).

Figure 187: Inmate information displayed on the InmateSearch screen

- 9 Click the **DC Number** link (Figure).
COTAS retrieves the standard inmate information from the existing inmate management system (Figure).

Figure 188: Standard inmate information from the Corrections Offender Information Network

Evaluating risk factors with the Facility Predictor

Like the Inmate Predictor, the Facility Predictor begins at the **RegionPredictorDashboard** screen (Figure). With the Facility Predictor, you can investigate a facility's relative risk of violence.

Follow these steps to start the Facility Predictor:

- 1 In the bottom row of the **RegionGuages** screen, under **Future Predicted Violent Events Based on Inmates** (Figure), click the region graph or link (**Region 3** in this case).
*COTAS displays the **RegionPredictorDashboard** screen for Region 3 (Figure).*

Figure 189: Facility Predictor image and link on the RegionPredictorDashboard screen

- 2 Under **Region 3 Predictors**, click **Facility Predictor** graph or link (Figure).
*COTAS displays the **RegionPredictorList** screen (Figure).*

Figure 190: RegionPredictorList screen

Note: As shown in Figure , the **RegionPredictorList** screen shows relative amounts of the facility's violence risk with increasing amounts of red. Yellow represents less risk, but may be cause for concern. Green represents substantially less risk. Risk is based on gang activity at the facility relative to the population size.

- 3 Click the **Facility** link (in this case, the **TARPON SPRGS W.R.C.** link in Figure).
*COTAS displays the **FacilityPredictorSummaryByDorm** screen (Figure), which enables you to drill deeper into the details of the risk.*

Figure 191: Gang tab on the FacilityPredictorSummaryByDorm screen

Note: As with similarly structured screens, the tabs display the same information with the columns slightly rearranged and grouped by the item indicated by the name of the tab as shown in figures 192 and 193.

Figure 192: Dorm tab on the FacilityPredictorSummaryByDorm screen

Figure 193: Primary Work Assignment tab on the FacilityPredictorSummaryByDorm screen

- 4 Click the **Gang** link (in this case, **LATINKINGS** in Figure 193).

*COTAS displays the **GangDetail** screen (Figure 4), which provides information about the gang, their allies and their enemies.*

Figure 194: GangDetail screen